ABSTRACT


Software system design is commonly acknowledged to be difficult: the problems are ill-formed, complex, and lack complete information. Expert designers employ strategies developed and refined over years of experience, while novices attempt to apply methods they have learned in the classroom. Existing research on software design education is scarce and focused on identifying the characteristics of student designers versus the attributes of expert designers. I am interested in studying how to bridge this gap through the novel adaptation of a model of complex systems development and evolution. This adaptation is realized in the Principles, Patterns, and Process Framework ($P^3F$), a suite of learning tools designed to help software design students learn and apply key aspects of expert designers’ strategies and behaviors.

This dissertation presents an experimental validation of the $P^3F$ through a triangulated research approach. Collected data measured the research subjects’ perceptions of their design skills and strategies, identified and categorized actual design behavior observed while the subjects worked on complex software system design problems, and evaluated the quality of the solutions they produced for these problems. The results of this study suggest that the $P^3F$ can help a novice software designer adopt and apply expert-like design strategies, recognize the development of these strategies as part of their own design process, and contribute to an improvement in the quality of their designed solutions.

Three significant contributions are made by this dissertation. It provides a comparative analysis of expert and novice design strategies and behaviors in a form not found in the literature. It also demonstrates a triangulated experimental model that features three research methodologies and analysis techniques. Most importantly, it presents and validates the Principles, Patterns, and Process Framework.
© Copyright 2012 by David Richard Wright

All Rights Reserved

by
David Richard Wright

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2012

APPROVED BY:

Dr. Mladen A. Vouk
Dr. Edward F. Gehringer

Dr. John O. Tector
Dr. Thomas L. Honeycutt
Chair of Advisory Committee
DEDICATION

To my wife and best friend Sharon, for her love, support, and patience.
BIOGRAPHY

David was born on Sunday, October 16, 1960, in Sewickley, Pennsylvania, a suburb of Pittsburgh, just three days after Bill Mazeroski smashed a home run in the bottom of the ninth inning to give the Pittsburgh Pirates a victory over the New York Yankees in game seven of the 1960 World Series. This was the first, and to this day the only, World Series won with a game winning home run in Game 7. An auspicious beginning, to be sure! David lived in suburban Pittsburgh until June of 1971, when his father’s job as an electronics engineer with Trion Corporation caused his immigration to Sanford, North Carolina.

David got married in 1981 and became the proud father of two children over the next five years. His daughter, Leia (named after the princess of the Star Wars fame), graduated from UNC-Greensboro with a double major in Elementary Education and Sociology. After teaching for four years, she is pursuing a master’s in Counseling at DePaul University in Chicago. David’s son, Jonathan, attended Western Carolina University before enlisting in the Air Force. Recently promoted to the rank of Staff Sergeant, he has been helping to maintain the fleet of B-52 strategic bombers and is cross-training to become a flight engineer.

In 1997, David married Sharon McCarthy, a New Jerseyite by way of Florida. They met in a bowling alley, of all places. Their wedding was a intimate affair in Lake Tahoe, Nevada, less than two weeks before David became a freshman student at State, and just three months after she graduated from Meredith College with a degree in Sociology. The adjustment to full-time college student from full-time employment was difficult at times. David’s intent at that time was to earn a bachelor’s degree and return to the work force. Not long into that journey, however, David realized that the field of Computer Science held a wealth of new and exciting challenges, and that to be in a position to meet these challenges a graduate degree was necessary.

David completed his Bachelor of Science degree in 2001, followed in 2003 by a Master of Science in Computer Science. The continued journey towards this doctoral brought many new challenges and opportunities that helped to shape David’s vision of his future career. In addition to a variety of Teaching Assistant appointments and other teaching opportunities, David served for two years as the president of the Computer Science Graduate Student Association. In this role, he helped to revitalize the organization and address several issues important to the graduate students in the department.

Working as a Junior Fellow with the Land Grant University Research Ethics initiative and as a Walter Wilkinson Research Ethics Fellow, David contributed to expanding the awareness of ethical and responsible research conduct in Computer Science and Software Engineering. David was also participated in the NCSU Preparing the Professoriate program and earned a Certificate of Accomplishment in Teaching, both of which helped prepare him for a future academic career.
ACKNOWLEDGEMENTS

There are many people who deserve thanks for helping me achieve this milestone. I certainly would not have reached this point without the love, patience, and support of my wife, Sharon. I also have to thank my children Leia and Jonathan and Jonathan’s wife Cassy, as well as my parents Richard and Margery Wright for understanding and encouragement. I could not have made it without their support.

I also have to thank Tom Honeycutt, my advisor and committee chair, for over eight years of guidance, confidence, and intellectual stimulation that went far beyond the subject of this work. Tom always seemed to find the right ways to energize my creativity and sharpen my thoughts when I needed it most. Paraphrasing Robert Frost, he encouraged me to take the road less traveled, that that made all the difference!

I would be remiss not to thank the members of my dissertation committee whose unique contributions made sure I got here. Mladen Vouk always seemed to cross my path when I needed a fresh perspective on a difficult situation, providing just the impulse necessary to keep my work on the right track. John Tector challenged and expanded my thinking about the importance of the human aspects of design and was invaluable in helping me articulate where this research fits into the larger picture of design research. Ed Gehringer joined my committee late, filling an unexpected vacancy and offering a practical, down-to-earth view that I had come to expect through the other opportunities I have had to work with him over the years. Thank you all!

I am also grateful for the financial support for this research that was provided by Ronald Holanek and the Financial Risk Group, Inc. and the NCSU Computer Science Department.

There are many, many others that have contributed to the achievement of this goal: teachers, professors, fellow students, university staff, and friends. To each of you I say, “Thank You!”

Trademark Acknowledgements:

- Atlas.ti is a trademark of ATLAS.ti Scientific Software Development GmbH.
- Eclipse Helios is a trademark of the Eclipse Foundation, Inc.
- Java, JavaScript, JEE, MySQL, MySQL Workbench, and OpenOffice are trademarks of Oracle and/or its affiliates.
- Access, Excel, PowerPoint, Visio, Visual Studio, and Word are trademarks of Microsoft Corp.
- PHP is a trademark of The PHP Group.
- Python is a trademark of the Python Software Foundation.
- StarUML is a trademark of Plastic Software, Inc.
TABLE OF CONTENTS

List of Tables ........................................................................................................................................ vii
List of Figures ......................................................................................................................................... viii

Chapter 1 Introduction .......................................................................................................................... 1
  1.1 Research Questions and Goals ....................................................................................................... 2
  1.2 Overview of Research Study and Results ...................................................................................... 4
  1.3 Organization of Dissertation .......................................................................................................... 4

Chapter 2 How Do Designers Design? ................................................................................................... 6
  2.1 What is Design? ............................................................................................................................. 8
  2.2 Strategies of Expert Designers .................................................................................................... 13
    2.2.1 How Experts View the Design Problem ................................................................................. 13
    2.2.2 How Experts Manage and Navigate the Design Space ....................................................... 16
    2.2.3 How Experts Make Design Decisions ................................................................................... 18
  2.3 Strategies of Novice Designers ..................................................................................................... 20
    2.3.1 How Novices View the Design Problem .............................................................................. 21
    2.3.2 How Novices Navigate the Design Space .............................................................................. 22
    2.3.3 How Novices Make Design Decisions ................................................................................ 23
  2.4 Pilot Study Results ......................................................................................................................... 24
    2.4.1 Pilot Study Results and Conclusions .................................................................................... 28
  2.5 Adapting Expert Strategies to Help Novice Designers .................................................................. 32
    2.5.1 Summary & Comparison of Expert and Novice Design Behaviors ....................................... 33
    2.5.2 Insights from the Pilot Study ................................................................................................. 34
    2.5.3 Suggestions for Improving Design Skills ............................................................................ 36

Chapter 3 The Principles, Patterns, and Process Framework .................................................................. 39
  3.1 Characteristics of a Framework to Support Novice Software Designers ....................................... 41
  3.2 Foundational Software Design Principles .................................................................................... 42
  3.3 The Principles, Patterns, & Process Framework ............................................................................ 45
    3.3.1 Principles for Relationships and Values ................................................................................ 48
    3.3.2 Patterns of Connections and Meanings ............................................................................... 76
    3.3.3 A Process for Decision-making and Movement .................................................................. 77
    3.3.4 The Decision Pattern .......................................................................................................... 88
    3.3.5 Using the \( \mathcal{P} \) ............................................................................................................... 93
  3.4 Summary ......................................................................................................................................... 103

Chapter 4 Research Methodology ......................................................................................................... 107
  4.1 Research Methodology .................................................................................................................. 108
    4.1.1 Methods for Observing Behavior ......................................................................................... 110
    4.1.2 Methods for Self-Assessment of Design Skills ................................................................. 112
  4.2 Research Study Design and Implementation ................................................................................... 113
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Design Decision Coding Matrix</td>
<td>26</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Common Decision Sequence Labels</td>
<td>27</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Identified Novice Design Behaviors</td>
<td>28</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>Identified Expert Design Behaviors</td>
<td>29</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>McNemar Test Statistics for All Subjects</td>
<td>31</td>
</tr>
<tr>
<td>Table 2.6</td>
<td>McNemar Test Statistics for Graduate Students</td>
<td>32</td>
</tr>
<tr>
<td>Table 2.7</td>
<td>McNemar Test Statistics for Undergraduate Students</td>
<td>33</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Fundamental Software Design Principles</td>
<td>43</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Fundamental Axioms of Software Architecture Design</td>
<td>44</td>
</tr>
<tr>
<td>Table 3.3</td>
<td>Fundamental Principles of Software Architecture Design</td>
<td>45</td>
</tr>
<tr>
<td>Table 3.4</td>
<td>Principles of Object-Oriented Design</td>
<td>46</td>
</tr>
<tr>
<td>Table 3.5</td>
<td>Design Principle Pattern elements</td>
<td>78</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Research questions and possible success criteria</td>
<td>108</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Design problem set assignments</td>
<td>119</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Decision Level Code Values</td>
<td>132</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Decision Type Code Values</td>
<td>133</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Decision Rationale Code Values</td>
<td>134</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>Non-design Codes</td>
<td>135</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>Inter-rater reliability of verbal protocol coding</td>
<td>136</td>
</tr>
<tr>
<td>Table 4.8</td>
<td>Inter-rater reliability for design artifact evaluations</td>
<td>142</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Second-level code scheme for protocol analysis</td>
<td>147</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Mean evaluations for design problems</td>
<td>148</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Exit survey responses related to understanding the $\mathcal{P}^3\mathcal{F}$</td>
<td>208</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>Exit survey responses related to $\mathcal{P}^3\mathcal{F}$ web site</td>
<td>215</td>
</tr>
<tr>
<td>Table E.1</td>
<td>Non-design Coding</td>
<td>252</td>
</tr>
<tr>
<td>Table E.2</td>
<td>Decision Level Coding</td>
<td>253</td>
</tr>
<tr>
<td>Table E.3</td>
<td>Decision Type Coding</td>
<td>255</td>
</tr>
<tr>
<td>Table E.4</td>
<td>Decision Rationale Coding</td>
<td>256</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Routine, innovative, and creative design spaces</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Graphical view of $P^3F$ relationships</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>Interactions of the Fifteen Properties</td>
<td>52</td>
</tr>
<tr>
<td>3.3</td>
<td>Levels of Scale – Mosque portico</td>
<td>55</td>
</tr>
<tr>
<td>3.4</td>
<td>Strong Centers – Angkor Wat</td>
<td>57</td>
</tr>
<tr>
<td>3.5</td>
<td>Boundaries – Tile art</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Alternating Repetition – Fern Leaf</td>
<td>59</td>
</tr>
<tr>
<td>3.7</td>
<td>Positive Space – Blue Nude</td>
<td>61</td>
</tr>
<tr>
<td>3.8</td>
<td>Good Shape – Chladni plate</td>
<td>62</td>
</tr>
<tr>
<td>3.9</td>
<td>Local Symmetries – Canal houses</td>
<td>64</td>
</tr>
<tr>
<td>3.10</td>
<td>Deep Interlock &amp; Ambiguity – Dovetail joint</td>
<td>66</td>
</tr>
<tr>
<td>3.11</td>
<td>Contrast – Stone garden</td>
<td>67</td>
</tr>
<tr>
<td>3.12</td>
<td>Gradients – Nautilus shell</td>
<td>68</td>
</tr>
<tr>
<td>3.13</td>
<td>Roughness – Inca Wall</td>
<td>70</td>
</tr>
<tr>
<td>3.14</td>
<td>Echoes – Native basket</td>
<td>71</td>
</tr>
<tr>
<td>3.15</td>
<td>The Void – Saint Peter’s Square</td>
<td>73</td>
</tr>
<tr>
<td>3.16</td>
<td>Simplicity &amp; Inner Calm – Shaker Cabinets</td>
<td>74</td>
</tr>
<tr>
<td>3.17</td>
<td>Not-Separateness – Stone path</td>
<td>76</td>
</tr>
<tr>
<td>3.18</td>
<td>Reasoning in the Design Cycle</td>
<td>80</td>
</tr>
<tr>
<td>3.19</td>
<td>Triggers of Formal Generation</td>
<td>81</td>
</tr>
<tr>
<td>3.20</td>
<td>External factors affecting the software design process</td>
<td>82</td>
</tr>
<tr>
<td>3.21</td>
<td>Simple argument map</td>
<td>89</td>
</tr>
<tr>
<td>3.22</td>
<td>Design Rationale in the design process</td>
<td>90</td>
</tr>
<tr>
<td>3.23</td>
<td>A $2 \times 2$ matrix for solving Pascal’s Wager</td>
<td>91</td>
</tr>
<tr>
<td>3.24</td>
<td>The Decision Pattern</td>
<td>92</td>
</tr>
<tr>
<td>3.25</td>
<td>Using the $P^3F$: Iteration 1 artifacts</td>
<td>95</td>
</tr>
<tr>
<td>3.26</td>
<td>Using the $P^3F$: Decision 1 documentation</td>
<td>96</td>
</tr>
<tr>
<td>3.27</td>
<td>Using the $P^3F$: Iteration 2 artifacts</td>
<td>97</td>
</tr>
<tr>
<td>3.28</td>
<td>Using the $P^3F$: Decision 2 documentation</td>
<td>98</td>
</tr>
<tr>
<td>3.29</td>
<td>Using the $P^3F$: Iteration 3 artifacts</td>
<td>99</td>
</tr>
<tr>
<td>3.30</td>
<td>Using the $P^3F$: Decision 3 documentation</td>
<td>100</td>
</tr>
<tr>
<td>3.31</td>
<td>Using the $P^3F$: Iteration 4 artifacts</td>
<td>101</td>
</tr>
<tr>
<td>3.32</td>
<td>Using the $P^3F$: Decision 4 documentation</td>
<td>102</td>
</tr>
<tr>
<td>3.33</td>
<td>Using the $P^3F$: Iteration 5 artifacts</td>
<td>103</td>
</tr>
<tr>
<td>3.34</td>
<td>Using the $P^3F$: Decision 5 documentation</td>
<td>104</td>
</tr>
<tr>
<td>3.35</td>
<td>Typical novice approach to the pothole tracking system problem</td>
<td>105</td>
</tr>
<tr>
<td>3.36</td>
<td>Design Methods: What the $P^3F$ is not</td>
<td>105</td>
</tr>
<tr>
<td>3.37</td>
<td>Design Thinking: What the $P^3F$ is</td>
<td>106</td>
</tr>
<tr>
<td>4.1</td>
<td>$P^3F$ web page layout</td>
<td>116</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Well-developed design decision-making skills are important for competent software analysts and designers, yet it has been noted that formal education is generally inadequate for developing these abilities in novices\[210]. Part of this problem is due to the time available within a student’s course of study, where strong competition exists with other subjects like algorithms, data structures, programming, databases, etc. Another cause is related to the difficulty of the subject of design itself: to learn to design, the student must be presented with complex, open-ended problems for which there may be more than one “correct” solution. Constructing these problems and evaluating students’ performance, in terms of how solutions are obtained and in terms of the correctness of those solutions, is difficult and time-consuming. Boulangera and Smith note that industry needs engineers and designers that can be immediately and productively effective\[33].

As more and more of our world becomes dependent upon computer-based systems, future software developers and designers must develop effective decision-making skills and strategies in addition to the technical knowledge they acquire through their education.

The decision-making strategies commonly taught in Computer Science/Software Engineering curricula and employed by novice software developers/designers are not effective when applied to large, complex real-world problems. Student (novice) designers tend to think linearly and concentrate only on the problem at hand\[52]. Novices also tend to use trial and error strategies and lack confidence in their design decisions\[9]. The production of artifacts (e.g., programs, UML diagrams, etc.) often takes precedence over the design decision-making processes used to create these artifacts. The problems they are asked to solve are usually compact, well-formed, and have a small set of “correct” solutions\[109]. Performance is measured by how closely the student’s work match the desired outcomes and specifications. How the student developed the artifacts is generally not a concern: if the “correct” solution was delivered, the student must have used the “correct” design approach.

While these skills may be sufficient for translating thoroughly documented designs into
working code, they are inadequate for reasoning about the large and complex software systems that the modern world depends upon[85]. When faced with large and ill-defined problems, the linear and deductive software design approaches commonly presented in software design and engineering textbooks break down[52, 131]. Students of software design and engineering need tools to help them learn, develop, and apply expert design strategies to help them mature into productive software design professionals.

The Principles, Patterns, and Process Framework (P³F) has been developed in response to this need. The P³F is a unique synthesis of results and ideas from a wide range of disciplines that provides novice software designers with a simple yet rich and flexible guide to help them quickly cultivate the key processes and behaviors of expert designers. This review will present the multi-disciplinary and foundational work that is brought together in the structure and content of the P³F in support of the overarching research question answered by this dissertation: How can software design students develop effective design decision-making strategies like those used by expert designers?

1.1 Research Questions and Goals

The purpose of this research is to create a learning framework to help software design students adopt and apply expert design decision-making strategies. Thus, the general research question can be stated:

*How can software design students develop effective design decision-making strategies like those used by expert designers?*

To answer this question, additional research questions also must be answered, including:

- What are the key strategies employed by expert designers to solve complex design problems?
- What are the critical weaknesses in the approaches that novices typically use when attempting to solve difficult design-oriented problems?
- Which of the identified expert strategies would directly address the critical weaknesses of novice design approaches?
- How can these expert strategies be incorporated into a novice-oriented knowledge-based learning and application framework?
- Using the created framework:
  - Are software design students able to understand the basic structure and application of the framework and the decision-making process it incorporates?
– Are software design students able to demonstrate the expert strategies incorporated into the framework when presented with a complex, ill-structured design problem?
– Does the framework improve the quality of design artifacts produced by the student?
– Does the framework improve the student’s perception of his or her design abilities and enable them to be more confident in their design decision-making?

In order to judge the success or failure of the proposed research, these research questions must be restated as research goals and identifying measurable success criteria for each goal. The \( \mathcal{P}^3 \mathcal{F} \) is proposed as a solution to the overarching research question and the subsidiary questions are stated to lead to this conclusion. We begin by restating the principle research question as a research goal:

The overall goal of this research is to validate the Principles, Patterns, and Process Framework as a tool for helping software design students learn and use expert design strategies.

To achieve this goal, eight objectives must be met:

1. Through a review of prior research, identify common strategies and behaviors of expert designers, including software designers as well as design experts from other design-intensive fields such as architecture, industrial design, mechanical engineering, etc.

2. Through a review of prior research, identify common strategies and behaviors of novice designers. As with studies of expert designers, this review will incorporate studies in the fields of computer science, software engineering, and information systems development and other design-intensive disciplines.

3. Through a synthesis of the reviews of prior research, identify weaknesses in novice design approaches and the corresponding expert strategies and behaviors that could resolve them.

4. Incorporate these expert strategies and behaviors into a knowledge-based learning and application framework that is accessible to novices.

5. Demonstrate that software design students are able to understand the basic structure and application of the framework and the decision-making process it incorporates.

6. Demonstrate that software design students apply the expert strategies incorporated into the framework when attempting to solve complex, ill-structured design problems.

7. Demonstrate an improvement in the quality of designed artifacts produced by students using the \( \mathcal{P}^3 \mathcal{F} \).
8. Demonstrate that a student’s perception of his/her design abilities and confidence in their design decision-making improves as a result of using the $P^3F$.

Meeting these objectives will make several important contributions to the software design body of knowledge. This work will provide a comparative analysis of expert and novice design strategies and behaviors in a form not found in the literature. It will also provide an experimental model that features three research methodologies and analysis techniques. Most importantly, it presents and validates the Principles, Patterns, and Process Framework.

1.2 Overview of Research Study and Results

The overarching experimental design employed in this research study is a interrupted time series quasi-experiment with two observations before and after the experimental treatment. One-group pretest-posttest studies are commonly used to identify changes that occur in a population as a result of some treatment or intervention, particularly in research involving learning and other cognitive processes\[42, 225]\]. Within this experimental framework, three different research methods are used to collect distinctly different types of data.

Primary data collected during the study were self-assessment questionnaires completed by the subjects at each observation session, audio-video recordings of the problem-solving episodes, and the paper and/or electronic artifacts representing the subjects’ solutions to the given design problems. Self-assessment questionnaire responses were analyzed directly for each subject in order to determine what, if any, changes in the subjects’ perceptions of their own design strategies and behaviors occurred after they were exposed to the $P^3F$. The audio/video recordings were analyzed using verbal protocol analysis, again examining the data for each subject independently to identify any behavioral changes that might have been triggered by the experimental treatment. The solution artifacts were evaluated for their quality based on a standardized rubric, and the results of these evaluations were then analyzed to discover any changes that could be explained by the introduction of the $P^3F$.

The results of the analysis provide evidence that one subject did perform at a level more like an expert than a novice after being introduced to the $P^3F$. Two other subjects demonstrated more modest improvement, and the remaining two subjects showed no change at all.

1.3 Organization of Dissertation

This dissertation reports on the research study that was conducted to investigate and validate the Principles, Patterns, and Process Framework. Chapter 2 presents a comparative review of the literature of design, focusing on identifying the key traits of expert and novice designers. The
\( \mathcal{P}^3 \mathcal{F} \) is introduced in Chapter 3, followed by a detailed discussion of the research methodology in Chapter 4. The specifics of the data analysis techniques and the results obtained from the analysis are presented next in Chapter 5. This report concludes with Chapter 6, which summarizes the conclusions and contributions of this work, identifies the limitations of this research study, and identifies future directions for research based on this study.
Chapter 2

How Do Designers Design?

The body of research pertaining to learning to design software systems is scarce, with most research efforts studying expert designers and/or comparing novices and experts. Intending to develop cognitive models of design behaviors, these studies consider novices and experts to be oppositional entities and do not attempt to explain how a novice develops the skills and knowledge to become an expert. This situation is not unique to software design research: studies of experts and novices in engineering design and architecture also fail to address the learning processes associated with developing design expertise. Other than identifying and describing behaviors of expert and/or novice designers, such studies offer little advice to aid in the transition from novice to expert.

However, the prior research into expert and novice design behaviors provides a rich and varied set of data necessary to answer the first three research questions and objectives identified in the previous chapter. Answering these questions will also provide the foundation for the development of the Principles, Patterns, and Process Framework which will be covered in the following chapter.

1. Through a review of prior research, identify common strategies and behaviors of expert designers, including software designers as well as design experts from other design-intensive fields such as architecture, industrial design, mechanical engineering, etc.

   What are the key strategies employed by expert designers to solve complex design problems?

2. Through a review of prior research, identify common strategies and behaviors of novice designers. As with studies of expert designers, this review will incorporate studies in the fields of computer science, software engineering, and information systems development and other design-intensive disciplines.
What are the critical weaknesses in the approaches that novices typically use when attempting to solve difficult design-oriented problems?

3. Through a synthesis of the prior research, identify weaknesses in novice design approaches and the corresponding expert strategies and behaviors that could resolve them.

Which of the identified expert strategies would directly address the critical weaknesses of novice design approaches?

This review of the literature identified three broad strategic areas that have been common subjects of study or highlighted in the results of studies of both expert and novice designers. How a designer views the design problem at hand is intimately connected to how the designer attempts to find a solution to the problem. Equally important is how designers manage their own knowledge of the design problem and use it to manipulate and navigate through the solution they are developing. The third strategic area identified deals with the way the designer makes design decisions, including what information is used and how the decisions are grounded or justified. This review will show the clear differences between novice and expert strategies in these areas and use those differences to establish a foundation for the Principles, Patterns, and Process Framework.

A key difference between software system design and other design-intensive fields is that the products of software design generally do not lend themselves to the same kind of close examination that designed artifacts like buildings, bridges, automobiles, and other physical entities allow. As a result, studies of expert and novice software designers are relatively scarce compared to other disciplines such as architecture, mechanical engineering and other design fields that produce physical artifacts. Furthermore, many of these disciplines are considerably older than computers and software, and offer rich and varied views on design and designers. It has been noted that there are significant similarities in design across domains as diverse as software design, architectural design, naming, and letter writing[242]. For this reason, I have extended the literature examined in this review to include a variety of studies across a broad spectrum design-intensive disciplines.

This review begins in Section 2.1 with a discussion of the terms design and designer to establish clear definitions of these and related terms along with a comparison of software system design with other design-intensive disciplines represented in this review. Section 2.2 reviews studies of expert designers to identify specific strategies, behaviors, and approaches used by experts and to categorize them in the three strategic areas noted above. Similarly, studies of novice designers are reviewed in Section 2.3. A pilot study was undertaken to investigate the key elements of the \( P^3F \) and to ground the current research project. The results of this pilot study are discussed in Section 2.4. This chapter concludes with a synthesis comparing expert
and novice behaviors, insights gained from the pilot study, and pedagogical advice from the literature regarding how to improve novices’ design skills in Section 2.5. This synthesis will be used in the next chapter as a basis for the design of the Principles, Patterns, and Process Framework.

2.1 What is Design?

Before considering the design-related behaviors of experts and novices, an important question must be considered: “What is design?” There are many different definitions of design based on the different perspectives and motivations of the definers. A clear definition is necessary to guide the investigation of prior design research leading to an understanding of the characteristics of novice and expert designers. The objective of this section is to examine these different perspectives and definitions and to develop a guiding view of design.

In the software engineering vocabulary, design is the iterative analytic transformation of the requirements and specifications for a product into a working system[201, 204, 220]. Software design is that part of a software development project that occurs after the development and refinement of requirements, and specifications and before program code is written and tested[30, 157, 214]. This analytical transformation is a rational approach to resolving the problem described by the specifications and requirements. However, this definition (and the approach it implies) does not capture the underlying human processes that make the transformation from requirements to code possible. In particular, this view does not take into account ill-formed and/or poorly understood problems, situations where analytical methods are rarely sufficient to produce a viable solution. Furthermore, when design is viewed only as a component in the software life-cycle rather than a pervasive activity throughout the development process, aesthetics and usability become far less important than the correctness of the design relative to a given set of specifications[142].

There are many other views and definitions of design and designing. Schön asserts that good designing involves a reflective conversation between the designer and the situation[221]. Reflective practice is a means of learning and internalizing tacit knowledge, well-suited for uncertain, unstable situations - i.e., solving “wicked problems”. Reflective practice also provides a bridge between the artistic aspects of design and the scientific and methodological foundations[56]. Simon proposed that most applied scientific disciplines are concerned with artificial rather than naturally-occurring phenomena and that these artificial phenomena are the result of human design[229](emphasis added). In the introduction Design Methods: Seeds of Human Futures, Jones lists a variety of one-line definitions of design from a number of design philosophers[145].

“A goal-directed problem-solving activity.”
“Simulating what we want to make (or do) before we make (or do) it as many times as may be necessary to feel confident in the final result.”

“The conditioning factor for those parts of the product which come in contact with people.”

“Engineering design is the use of scientific principles, technical information and imagination in the definition of a mechanical structure, machine or system to perform pre-specified functions with the maximum economy and efficiency.”

“Relating product with situation to give satisfaction.”

“The performing of a very complicated act of faith.”

“The optimum solution to the sum of the true needs of a particular set of circumstances.”

“The imaginative jump from present facts to future possibilities.”

“A creative activity — it involves bringing into being something new and useful that has not existed previously.”

These definitions reflect a variety of perspectives on design, but the common theme they all address is the process of design rather than the product. Several of these definitions also reflect the creative nature of design: the act of bringing something into existence that has not existed in that particular configuration. Jones offers a more complete definition of design and the design process, casting design as not just a problem-solving activity, but a goal-oriented process of exploring, learning, and creating:

*The fundamental problem is that designers are obliged to use current information to predict a future state that will not come about unless their predictions are correct. The final outcome of designing has to be assumed before the means of achieving it can be explored: the designers have to work backwards in time from an assumed effect upon the world to the beginning of a chain of events that will bring the effect about.*[145]

Another perspective on design identifies three characteristic design situations and the types of design tasks that each motivates. Design activities are commonly classified into routine, innovative, and creative[35, 70]. Routine designs occur when the design problem and solution spaces are well-defined and well-understood, leaving the design task to be one of instantiation: how to combine and configure elements or entities existing in problem domain into a solution. Innovative designs also occur within a well-defined space, but result in familiar yet novel and different forms, new subtypes of existing general forms. Creative design tasks are non-routine,
a result of poorly defined or understood situations that require new insights and perspectives. Creative design decisions generate new types, configurations of new (and possibly some already existing) elements that extend or move the design space and may result in a paradigm shift within the domain of the design problem. Figure 2.1 illustrates the relationships between these types of designs[113].

![Figure 2.1: Routine, innovative, and creative design spaces](image)

A simple example of a routine design would be the decision to use a fixed-length array to store a finite data set in a software application. Extending a “built-in” class or type to provide behaviors and/or attributes optimized for a particular application would be an example of an innovative design decision. One example of this kind of design might be the development of aspect-oriented programming language constructs to help manage attributes and behaviors of program elements that cut across traditional architectural boundaries within a software system. Basically, there are two approaches to creative design: 1) start with existing elements and modify them to produce elements that did not exist before; 2) configure new elements from basic building blocks (first principles)[113]. It should be noted that while these three characterizations of design tasks are commonly agreed upon, there is some debate over the demarcation between innovative and creative design tasks.

Routine design tasks are often accomplished with analytical methods such as decomposition, transformation, and constraint satisfaction since they deal with well-defined design spaces and existing knowledge[18, 172, 119]. These are the types of situations computer science and software engineering students commonly experience in their post-secondary education. Because the design space is well-defined, innovative solutions may be derived through exhaustive searches
of untried combinations. Innovative design tasks also occur when the well-defined design space is manipulated in nonroutine ways, often driven by new objectives [78], the early generation of possible solutions [130, 38] and development of the problem space [74]. Problems that are ill-structured, poorly defined, and have a high degree of uncertainty, and as such are not amenable to solving using analytical methods [229, 69] and require creative approaches to generate possible solutions. It is in these situations that the experientially-developed strategies of expert designers stimulate alternative solutions outside the known design space. The analytic methods taught to and employed by novices are insufficient for these “wicked” problems.

These definitions of design and insights into the design process demonstrate the complexity of the design process, whether it is viewed as a methodological process or as a cognitive action. Above all, design is a human-centric activity, particularly when we consider the creative aspects of design. For this work, the following concepts are used to characterize design:

- Design is intentional and purposeful.
- Design begins in the mind of the designer as a product of his or her understanding of the problem, goals, constraints, requirements, etc.
- Design produces artifacts that exist in and interact with the real world.
- Design is a generative process that creates entities that did not exist before.
- Design is an iterative process that refines and integrates.
- Design is a social, reflective, and communicative process between the designer, the co-evolving problem and solution, and the stakeholders in the system being created.
- Design involves exploration of goals and alternatives and their possible combinations.
- Design involves learning as new aspects of the problem, solution, and context are unfolded during the design process.
- Design activity occurs in two general contexts: the designer’s operational environment and the environment produced by the developing solution. The designer’s perception of the design tasks drives shifts between these contexts.

Developing the analytic and creative skills in software designers is difficult. As Brooks explains:

*We can get good designs by following good practices instead of poor ones. Good design practices can be taught... Nevertheless, I do not believe we can make the next step upward in the same way. Whereas the difference between poor conceptual designs*
and good ones may lie in the soundness of design method, the difference between good designs and great ones surely does not. Great designs come from great designers. Software construction is a creative process. Sound methodology can empower and liberate the creative mind; it cannot inflame or inspire the drudge... Study after study shows that the very best designers produce structures that are faster, smaller, simpler, cleaner, and produced with less effort. The differences between the great and the average approach an order of magnitude.[34]

Kazmierczak shifts the emphasis away from the objects and artifacts of the design process and onto the meaning that is created and communicated by those artifacts[150]. Rust compliments this definition of design by noting that the artifacts of design must embody the tacit knowledge of the designer in a manner that makes that knowledge accessible to the receiver[217]. Jones states that the process of designing is “the designer’s way of discovering what he knows, and what he does not know, about this new thing he has promised to invent, and to integrate it into the world as it is.”[144] Knowledge acquisition, sharing, and integration are critical activities in the design of large software systems[250].

However, the process of designing is not only logical and rational, it is also creative and intuitive[221, 78, 144]. It is here that the connection with critical thinking begins to emerge. Quellmalz identified six key strategies of critical thinking, employed in a dynamic and metacognitive process that includes planning, monitoring, reviewing, and revising[25]:

- Identifying and defining issues and problems
- Determining the kind of information that is relevant to solving the problem or resolving the issue
- Gathering, judging, and connecting the information
- Generating hypotheses, constructing arguments, making inferences
- Testing hypotheses, making counter-arguments
- Evaluating the results, possibly revisiting earlier stages of the process

Design problems are usually “ill-structured”[228] and even “wicked”[58, 209] because they rarely have precisely prescribed goals or objectives, and the means of solving the (apparent) problem is not obvious[213, p.40-41]. Compound this with constantly changing technology and user requirements, and software system design becomes a very difficult problem. Software designers must continually refine and reformulate the problem(s) at hand, searching out new knowledge that may be applicable to the situation at hand, proposing alternative solutions based on the new and existing knowledge and experience, and learning from the evaluation of these proposed solutions[159]. The ability to objectively look at a problem from many different
perspectives and to formulate and evaluate potential solutions that integrate those different perspectives is a necessary quality of the best software designers[189].

Against this background of design and designing, prior research into the design process can be examined. In particular, this review will compare and contrast the strategies, approaches, and behaviors of experts and novices in order to identify specific aspects of experts that can be made accessible to novices in order to help them jumpstart their creative design skills and accelerate their growth and development as software professionals.

2.2 Strategies of Expert Designers

The strategies, behaviors, and processes used by experienced designers have been frequent subjects of study. The principal goal of these studies has been to thoroughly understand expert strategies in order to build models that can be used to implement “expert systems” to support the work of designers in specific fields. Common questions that are asked in this research include:

- How does the expert designer view a design problem?
- How does the expert designer manage and manipulate knowledge, information, tasks, and artifacts within the design problem space?
- How does the expert designer make decisions about the design problem?

Software design differs from other design fields in its relationship with the physical world. Most designers create artifacts that will have some type of physical form: a building, automobile, assembly line, etc. The result of designing a software system is a complex set of instructions specifying sequences of operations executed by some computational device to achieve some (usually non-computational) goal. Despite the difference in the medium of design, effective software designers apply much the same thought processes as their counterparts in other design fields[73]. Understanding the complete design problem, efficiently managing that understanding, and making reliable decisions about the design problem and process for solving it are all critical aspects of the expert designer’s cognitive toolkit. The remainder of this section examines the literature of design research to identify key features of these aspects, answering the questions above and defining the common strategies of expert designers.

2.2.1 How Experts View the Design Problem

How a design problem is envisioned by the designer is a critical aspect of how the solution to the problem will be developed. The ability to maintain a systemic or “big picture” view and understanding of the design problem within the application space where the problem occurs is a
critical skill exhibited by expert designers. This is often accomplished through the construction of abstract mental models, the use of different design languages to represent and communicate design information, and the use of visual analogies to help bridge known and unknown situations within a design task or problem[6, 51, 90].

Seeking to identify the “seeds of best practice” through the study of expert designers, Cross and Cross[75] examined the strategies of two outstanding designers. They deliberately chose complex, ill-defined problems rather than well-formed and routine problems in order to study how the designers approached more realistic problems. One of the strategies they discovered is the practice of maintaining a systemic view of the design problem and solution. This view considers not only the entirety of the developing solution itself, but also incorporates the application space of the problem/solution: where the solution will exist when it is implemented. The authors also compared their results to studies of “creative experts” and found similar parallels between their subjects and other creative designers, reinforcing the proposition that systemic viewing of the design problem is a widespread attribute of experts working on ill-formed and complex problems.

Dym, Agogino, Eris, Frey, and Leifer examine this strategy in greater detail, identifying four key aspects of systemic thinking[90]. The first of these, thinking about system dynamics, involves the ability to understand the interactions among the different parts of a system and to anticipate the unintended consequences that can emerge from these interactions. The ability of experts to reason about uncertainty and work with incomplete information and ambiguous objectives is the second critical aspect of systemic thinking in expert designers. As systems become larger and more complex, it becomes difficult for designers to manage all of the details of the design simultaneously. The aspect of systemic thinking commonly employed by experts is the ability to estimate the essential factors and selectively focus on them, filtering their view to what is necessary for the particular purpose at hand. The fourth aspect of systemic thinking identified is the ability to efficiently plan experiments as part of the design process and to understand the results of these experiments.

The ‘Limited Commitment Mode’ (LCM) control strategy is another model of the cognitive processes experienced designers use to maintain their view of a complex system. The LCM strategy allows a designer to put a module or piece of the design “on hold” while working on other parts of the system, even if they are not related — the designer is not required to complete each module before starting to work on another[115, 116]. Goel and Pirolli note that there are three possible contexts the designer must maintain: 1) the particular part of the system under consideration; 2) the whole of the system in its current, partially developed state; and 3) the projected complete state of the system as it will be used[116]. The third context is especially important as it represents the situatedness of the final design which the designer must keep in mind throughout the design process[114, 249].
Carroll further expands this notion of situated design by detailing five holistic contextual dimensions of the design process: domains of human activity, roles of stakeholders in the system, shared representations of the design, the interactions among the participants in the design and use of a system, and the relationships of users to the design process[46]. Business or domain experts understand their business and its problems (and inherent constraints), and the software system designer must capture this understanding in the solution requirements. The designer must also recognize the nonfunctional requirements representing capabilities that are orthogonal to domain requirements and that often surface as unstated customer expectations[178].

Another strategy used by expert designers to maintain the big picture view of the system is the use of abstractions and mental simulations. Abstraction may take the form of removing non-essential elements, attributes, etc., acting as a filtering mechanism[90]. Studying expert designers in familiar and unfamiliar domains, Adelson and Soloway found that experienced designers use “labels” to identify previously solved problems that are elements within the current design problem[6]. They also found that expert designers construct various mental models to represent and simulate relationships and interactions between known and unknown parts of a system.

Descriptive abstractions or metaphors are also used as placeholders for elements of a system that are not yet fully defined, but whose general properties or behaviors are known[24]. Metaphors are not intended to be literal statements of actual system elements or attributes, but easily understood substitutes that convey rich meaning in a concise package[231]. Abstractions serve to reduce the complexity of a complex system, but the level of detail and complexity of an abstraction must be selected with care: too much detail and the problem remains unmanageably complex; too little detail and the abstraction loses value as a model of the problem/solution[134, 161].

Experienced designers also understand that design problems are almost always poorly defined and that part of the design process must be dedicated to exploring and learning about the problem. As they explore and learn, designers must integrate this new knowledge into their understanding of the problem and the implications for the solution. The most common technique for learning about and exploring the design problem involves developing possible solutions based on what is currently known and evaluating those solutions to identify new unknowns. As they propose and evaluate possible solutions, they are often reminded of other issues that require consideration[155]. Noting challenges to the tradition model of design as two discrete phases (problem formulation and solution synthesis), Maher, Poon, and Boulanger developed a computational model of this co-evolution of problem and solution using genetic algorithms[173]. This model produced acceptable solutions to a well-defined designed architectural design problem, but also demonstrated instability similar to the unpredictability of human designers. Dorst and Cross confirmed this model in the processes of experienced industrial designers, noting that it is useful for modeling the ‘problem-solving’ aspects of design but does not accommodate
the creativity associated with human experts\cite{88}. Harfield\cite{125} extends the problem-solution concept by differentiating two categories of problem: the ‘problem as given’ and the ‘problem as design goal.’ The designer’s personal preferences, expectations, and prejudices not only affect how the designer views the “actual” problem, but also constrain how it can be solved.

As noted earlier, software systems designers work in an environment that is fundamentally different than that of most other design-related disciplines as they are computational systems intended to solve problems that are often not described as computational problems. System designers and developers often are not experts in the application domains of the software they create. Exceptional software designers are able to build and maintain relationships between the application domain where the problem exists and the computational domain where the solution will be implemented\cite{76}. Additionally, software designers must be able to manage many different views of the problem/solution, including the business context and related processes, evolving technology and hardware resources, changing information used by the system, and the tools used to implement, deliver, and maintain the system\cite{100, 137, 156}.

2.2.2 How Experts Manage and Navigate the Design Space

The ability to effectively navigate in the software design space is also critical and is closely related to how the designer envisions that space. Several key skills that expert designers use to navigate and manage large, complex system designs include a high tolerance for ambiguity and uncertainty, the ability to “chunk” particular elements or aspects of a system for closer attention, creating and maintaining flexible heuristic plans for navigating systems, and the ability to think and communicate about the system in different representations and languages.

Peterson found that outstanding engineers and planners are characterized by open-mindedness and a high tolerance for ambiguity, traits that allow them to opportunistically move about in a design space in response to what is known and unknown at a given point in time\cite{121, 200}. Opportunistic should not be taken as random or uncontrolled. Cognitive studies of expert designers suggest that they employ schema-based, heuristically-driven, and well-connected mental planning structures to guide the exploration of the problem/solution space\cite{210}. These planning structures provide high-level control of the design process, enabling emergence of new details and activities not derivable from the activity being processed\cite{163}.

Carroll, Thomas, and Malhotra suggest a relationship between the cyclic planning structures of expert designers and the structure of the solutions, in particular that a cycle or move in the design space depends upon the results of the cycles or moves that preceded it. They also found that a move may also render earlier cycles unnecessary\cite{48}. Furthermore, as the structure of the designer’s mental planning process becomes more well-defined, the solution structure tends to converge\cite{49}. The content of the solution structure reflects the structure of the exploration
process and vice versa. As a result, the content and structure concurrently describe the state of a system at a given point in time[118].

Foundational to all of these skills is the ability to reflect on both the design problem and the process one is using to find a solution to the problem. Schön calls this process *reflection-in-action*, describing it as a conversation between the designer and the design situation involving the construction of the problem, the strategies the designer applies to find some part of the solution, and the history of the decisions the designer has made up to that point[221]. Each decision the designer makes has consequences affecting the final solution, and the designer must “listen” to these as if the situation were “talking back.” By reflecting on this conversation, the designer can discover new ideas and information that generate new ways to frame and navigate the design space. The design problem itself may also be illuminated from a different point of view, allowing the problem and evolving solution to be reinterpreted[213].

Dym, et. al., note that expert designers anticipate consequences even when dealing with uncertainty and ambiguity, often using estimates and generalities as intermediates in their understanding of the design problem. Experts employ both divergent and convergent thinking to explore the design problem and generate possible solutions[90]. Eris also found that the dialog between the designer and the design situation, including the stakeholders in the intended product, frequently involve questions that are divergent and lead to many possible unknown answers[99]. The purpose of this kind of dialog is to learn about the design situation from a conceptual perspective. This point of view provides a foundation for navigating the design space as well as for asking the convergent questions necessary to define and specify the problem and solution in a testable form. The implication of these findings is that expert designers generate uncertainty intentionally in order to provide paths of inquiry that guide the navigation of the design space.

The ability to flexibly pull out or “chunk” arbitrary elements of a design situation is also necessary for efficient design situation navigation. Large, complex systems defy complete understanding, although smaller pieces of such systems can be bounded and situated in a context within the larger system in order to identify possible solutions. Schön found this process of problem setting to be another basic reasoning technique expert designers use to navigate design situations[221]. Problem setting is a dynamic cognitive strategy that allows the designer to expand, contract, and reorient his or her focus within the design space. Jackson[138] proposes a formalization of this problem framing technique, gaining valuable graphical documentation of the design situation at the expense of the ability to quickly reorient the focus of the designer’s attention within the problem space. In particular, many expert designers tend to choose challenging and novel ways of framing design problems in order to find new and better solutions[75, 187].

Combined with continuous reflection and the use of generative inquiry, problem setting
provides a powerful tool for developing and maintaining plans and strategies for navigating design situations. Rather than rigidly applying a single methodology or approach, the expert designer remains flexible and adaptive, opportunistically using the information at hand to refine the solution and development path[120]. Intermediate forms and hypotheses represent the designer’s knowledge at a particular point in time and also expose new aspects of the problem and solution[163]. Expert designers also tend to be pragmatists, ready and willing to change their approaches to suit different problems and situations[55]. Rigid design methodologies severely constrain the designer’s ability to adapt or change their approach when faced with new information and understanding of the design problem.

Expert designers also use different languages and representations to aid in navigating the design space. Part of this is a result of the variety of stakeholders the designer has to communicate with: customers, end users, regulatory agencies, and the people who actually construct the design object. Social concerns compete with technical issues throughout the development and life of software systems[193]. Recent studies suggest that working with different forms of representation is also essential for effectively moving through a developing system design. Bilda and Gero found that sketching and other visual representations helped to mitigate limitations in designers’ working memory while also providing valuable cues about what part of the system should be attended to next[29]. These cues frequently contribute to the invention of important issues and/or requirements that drive the co-evolution of the problem space and the solution space in the design situation[239].

2.2.3 How Experts Make Design Decisions

The goal of design, in general, is to intentionally change the built environment in order to bring it closer to some desired state. This change is brought about by a sequence of design decisions that are made in the context of the dynamic and co-evolving design problem and solution spaces. Design decision-making is closely related to the content of the decision and the process used for making a particular decision cannot be clearly separated from the context in which that decision is made[227]. Expert designers decision-making styles are characterized by open-mindedness, a high tolerance for ambiguity, orientation to purpose, and a preference for ‘soft’ or subjective information[200].

Design decisions that generate and choose between possible design alternatives are ubiquitous in software development, and the impacts of these decisions are significant to the quality of the delivered systems[129]. Akin and Lin define design decisions as any and all intentional declarations of information as valid for the design problem at hand, a definition which allows for the inclusion of routine and casual decisions as well as non-routine and creative decisions[195]. This differentiation is important as it provides a means for highlighting decisions that are
critical to the progress of the design process. Novel decisions rarely follow directly from previous
decisions and assumptions, but mark and resolve critical cognitive bottlenecks in the development
of the design[195].

A related way of describing design decisions follows from the inherent goals and objectives
of the decision. A rational or normative decision is “one that conforms either to a set of
general principles that govern preferences or to a set of rules that govern behaviour. These
principles or rules are then applied in a logical way to the situation of concern resulting
in actions which generate consequences that are deemed to be acceptable to the decision
maker”[237]. Rational decisions strive to obtain an optimal selection from the set of possible
alternatives[257]. The second type of decision is the naturalistic decision, one that “connotes
situational behaviour without the conscious analytical division of situations into parts and
evaluation according to context-independent rules.”[103, 153] Naturalistic decision making is
manifested in dynamic conditions, embracing ill-defined tasks and goals and employing expert
judgement over mathematical computation of alternatives. Most importantly to the discussion
at hand is that naturalistic decision making has a goal of satisficing rather than optimizing, the
acceptance of a “good enough” solution instead of the best solution[258].

Attempting to develop a logic-based model of the design process that incorporates naturalistic
decisions, Galle introduces the concept of “degrees of preferability” as a basis for choosing
between alternatives based upon the relevant points of view, i.e., those perspectives that are
relevant in the current design state[110]. In situations described by a statement \( S \) and its
negation \( \neg S \), these degrees of preferability are described as:

**Required** \( S - S \) is preferable to \( \neg S \) for all relevant points of view.

**Good** \( S - S \) is preferable from most relevant points of view, although there may be
some where \( \neg S \) is preferred.

**Desirable** \( S - S \) is preferable from some, but not most, relevant points of view.

Galle defines a rational design decision as one where the decision-maker knows or believes
that a situation described by \( S \) is either 1) required; 2) good; or 3) desirable and \( \neg S \) is not
good[110]. This model of design decision-making provides a basis for explaining value judgements
of different degrees, as well as incorporating the context or design state into the decision-making
process. The relationship between the design context and the designer’s knowledge is critical to
effective decision-making. Context is task- or activity-oriented and provides the framework in
which design decisions are made[188]. The designer’s knowledge must be intersected with the
current context, as viewed by the designer, in order to contextualize that knowledge and make
it available to be applied to the decision-making process[203].

The importance of context and the expert designer’s understanding and use of the state of
the design and its larger context cannot be minimized. Boulangera and Smith[33] identified
eight styles of design decision-making that implicitly incorporate contextual information and understanding, grouping them in opposing pairs:

- **Retriever/Deriver** - The *retriever* is heavily influenced by previous designs and readily seeks out and reuses elements of previous design solutions, while the *deriver* is lightly influenced by precedent and derives preliminary design elements from rules of thumb, design basics, and first principles.

- **Generalizer/Detailer** - The *generalizer* remains at a general level of design longer while a *detailer* commits to a particular detailed solution earlier in the design process.

- **Visualizer/Verbalizer** - The *visualizer* is opportunistic and uses sketches, visual aids, and prototypes to develop and refine the understanding of the design problem. The *verbalizer* builds linear task lists and follows them hierarchically with little or no deviation from the planned course of action.

- **Lateral thinker/Extrapolator** *Lateral thinkers* are inspired by fields lateral to their own and seek to find new solutions through examining fundamentally related problems in divergent fields. *Extrapolators* extend the known limits of their own fields through the creative extension of existing artifacts into new application domains.

In Boulangera and Smith’s work, designers employing lateral thinking and extrapolation were associated exclusively with non-routine or naturalistic decisions. The retriever, generalizer, and visualizer were also strongly associated with non-routine decision-making. Expert designers rarely use only one of these strategies during a design session, and commonly employ up to four of them (one from each set) as they work on a particular design problem[33]. The application of different styles and approaches within an iterative design refinement process is a hallmark of expert design behavior as a designer exposes new aspects of the design problem[90]. Schön and Wiggins note that the visual aspect of designing is particularly important to making value comparisons and judgements as well as in identifying relationships between design elements, perspectives, and requirements[223].

Expert designers rely on a variety of knowledge sources when making design decisions. Cross and Cross studied two expert designers and found that they both relied heavily on “first principles” as a basis for making critical decisions[75]. While background and domain knowledge are important aspects of design decision-making[76], their use tends to be imprecise and directed towards framing and structuring the problem rather than directly making decisions about a particular solution[110].

### 2.3 Strategies of Novice Designers

Students are generally good problem-solvers but poor decision-makers. Exposure to expert strategies and decision-making processes has the potential to enhance and complement other types
of design education[33]. However, the emphasis of engineering design education is on algorithmic problem-solving skills while practitioners deal with complex, ill-defined, and open-ended problems that require decision-making skills and strategies that can contend with ambiguous, uncertain, and poorly-defined information[37].

The problems students are asked to solve are usually compact, well-formed, and have a small set of “correct” solutions[109]. Such problems lend themselves to logical, analytical solution development, and are much more easily graded for correctness and performance than open-ended problems with a large set of satisfactory solutions. As a result, students learn what they need to to achieve their desired grades[79]. While developing these analytical problem-solving skills may be sufficient for completing typical academic work, they are inadequate for preparing the future software developer for professional practice[85]. When faced with large and ill-defined problems, the linear and deductive software design approaches commonly presented in software design and engineering textbooks break down[52, 131].

Studies of novice software engineers and designers is extremely scarce, with most studies of novices focused on programmers in introductory computer science courses. These studies commonly examine the application of various methods and techniques to teaching and learning basic programming concepts, structures, and languages. Expanding the search for studies of novice designers to fields other than computer science and software engineering yielded more results, but the literature is still sparse compared to studies of expert designers. This section examines this multi-disciplinary literature in order to identify the common traits of novice designers and to gain a clearer understanding of the common characteristics of novice designers.

2.3.1 How Novices View the Design Problem

Novice designers view design problems much differently than their expert counterparts. In particular, novice designers have a strong tendency to think linearly and concentrate only on the problem at hand[52]. This problem-centric view coupled with a deductive problem-solving approach effectively places blinders on the novice designer[117]. Eckerdal and her colleagues suggest that this kind of behavior should be taken in context, since students are generally “trained” to solve problems rather than to design solutions[92]. Their academic assignments usually specify, with significant detail, the form, function, and presentation of the responses they are to generate. These results further suggest that novice software designers may not fully understand and appreciate the difference between a “program” and a “software system,” particularly where issues of communication and complexity are concerned.

Novices also tend to focus on the literal or surface features of a problem in order to fit what is known and what is unknown about the problem into their fact-based knowledge[26, 160]. The novice designer seeks to force a new design problem to conform to specific solutions they
understand and can duplicate[24]. The novice’s literal focus also exhibits itself as a rush to an embodiment or implementation because the novice cannot separate concepts and ideas from the implementation of those ideas[154]. This is a reflection of novice designers’ reliance on what is explicitly stated in the problem description and their limited ability to generate inferences and relationships. As Christiaans and Dorst note, many novices get stuck in the process of information gathering in order to fill in as many known variables as possible, substituting this process for actual design work[57].

This emphasis on the obvious and literal aspects of a design problem appear to be a symptom of a deeper deficiency in the skill set of novice designers. Studies of aerospace engineering students found that novices are unaware of the information they need most of the time. Even when they recognized the topic they need to know about, they lack the ability to formulate questions to gain the needed information[8]. Confirming Christiaans and Dorst, these students also spent considerable portions of their working time gathering information rather than actually designing a solution. Ahmed, Wallace, and Blessing suggest that a possible cause of this extensive information-gathering activity may be the inability to differentiate between important and less-important issues in the design problem[9].

In a study of mechanical engineers, Ehrlenspiel and Dylla found that weaker, inexperienced designers view the problem and potential solutions at a more superficial level and with less accuracy than their more experienced counterparts[94]. A superficial view of the design problem coupled with a lack of attention to the environment in which the solution will function is a frequent cause of errors and failures in software engineering student projects[140]. This superficial view of a design problem contributes to the novice attitude that understanding and formulating the problem is of little importance[6, 141]. As a result, inexperienced designers often generate relatively complete (at least in their own mind) solution concepts directly from the problem statement, and these initial solutions become the basis for justifying their final solutions[176]. The tendency of novices to view problems superficially is not limited to the domain of design, but is evident even in general problem-solving strategies. Novick confirmed that novice problem solvers fail to correctly identify analogical structural similarities when given problems with different surface features[192]. Additionally, novices frequently misidentify the underlying structure of problems that are superficially similar.

2.3.2 How Novices Navigate the Design Space

The shallow view of design problems typical of novices affects how they manage and navigate the accumulating knowledge about the problem and solution. One of the consequences of their superficial perspective is evidenced by their difficulty with or failure to scope the design problem. Sutcliffe and Maiden assert that a significant cause of poor performance in novice software system
analysts is the failure to adequately identify requirements, stakeholders, and other non-software influences on the design problem[238]. Students from a variety of other engineering disciplines also evidenced this behavior[22]. Comparing novices and experts in architectural design, Casakin reports that novices’ inability to identify and apply analogies in their design processes hindered their ability to identify constraints not explicitly stated in the design problem[50]. Likewise, novices often fail to identify important goals and objectives that are foundational to resolving design problems[176].

As discussed earlier, the use of abstractions, metaphors, and analogies is an effective expert strategy for managing the complexity of designing large systems of any type. The ability to think abstractly also enables the designer to shift focus within the evolving problem/solution space in order to navigate freely through the design. Novice designers’ lack of abstract thinking skills significantly restricts their ability to manage and navigate their designs effectively[20, 21]. As a result, their primary search and navigation method is top-down, depth-first decomposition of the problem[131]. This constricted strategy is further hampered by difficulties in integrating the decomposed pieces back into a complete and working solution[166].

When decomposition fails the novice designer, the common response is to apply a trial-and-error strategy, exhaustively searching through possible solutions until a working candidate is found[9, 176, 10]. Trial-and-error is not an efficient design strategy, particularly when large numbers of possible solutions are generated and implemented without an overarching plan or goal[94, 50]. Not only is this strategy inefficient from the software development perspective, generating large sets of alternatives and attempting to manage them mentally without the use of abstractions or metaphors can easily exceed the designer’s cognitive capacity, diminishing their design thinking abilities even more[149, 148]. Further aggravating this kind of weak and ineffectual design strategy is the failure to critically evaluate proposed solutions and hypotheses about the design problem and generally inconsistent design problem-solving approaches[6, 141].

2.3.3 How Novices Make Design Decisions

Very little research could be found that specifically examines how novices make design decisions. However, some studies of the general design approaches of novice/student designers offer insights into decision-making strategies that are extensions of how they view and navigate design problems. Most commentators on novice design strategies point to a lack of understanding of the cognitive aspects of designing (how to make design decisions) as a fundamental cause of novices’ poor design skills, though this is a bit of a circular argument: students have trouble making design decisions because they do not know how to make design decisions. However, Boulanger and Smith note that students are good problem-solvers even though they are poor decision-makers, a situation that is likely because they are “trained” to solve academic problems[33].
As discussed earlier in this section, novices tend to follow linear, deductive problem-solving approaches that may be effective for well-formed problems, but that are inadequate for ill-structured design problems[52, 131]. Unlike experts, novices attempt to avoid uncertainty and ambiguity, and seek a solid, unchanging foundation for their decisions whenever possible[19, 227]. Their literal view of the problem and attempt to fit it into their fact-based experience is a means of reducing this uncertainty and ambiguity[160, 26]. When they cannot eliminate uncertainty, novice designers lack confidence in their design decisions[9, 10].

Once a decision is made, novice designers find it difficult to revisit or reconsider the decision, particularly when such reconsideration would entail reversing it[33]. Novices’ “rush to implementation” is further exemplified by their expressed need to document their decisions in code rather than through other means of communication[154, 104]. Because they strive to base their decisions on what they believe to be literal or unchangeable facts and these decisions are often made concrete through implementation, novices are generally unable or unwilling to change decisions that have been made[227].

2.4 Pilot Study Results

The preceding discussion highlighted significant differences exist between the strategies used by expert designers and and those used by novices to approach design problems that are ill-structured, open-ended, incompletely defined, and full of apparently conflicting information. The Principles, Patterns, and Process Framework (P3F) is proposed as a solution to the problem of encapsulating expert design strategies in a learning tool that focuses attention on the design decision-making process. A pilot study was conducted to assess the validity of the P3F. The specific goals of this study were to determine if novice software designers could understand and apply the P3F to complex design problems and to determine if the use of the Framework produced any changes in behavior in the novice designers. The results of the pilot study have been previously reported[255, 254, 256], and are summarized in this section.

The pilot study was conducted with students in two different software design-related courses, one at the advanced undergraduate level (Software System Design) and the other a first-year graduate software engineering course. As part of both courses, students completed two software system design homework assignments, one before and one after the introduction of the P3F. Students were given a one-page problem statement and asked to develop a conceptual design for the desired system. Both of the assignments involved developing a new software application that would integrate with one or more existing software systems to extend and enhance the customer’s operations and were adapted from system design problems in a popular software engineering textbook[204].

The first assignment was given at the beginning of the semester, before any in-depth
discussion of software system design, in order to establish a baseline for the students’ design skills. Students were informed that the assignments were also to be used as research data. After the first assignment was submitted, the $P^3F$ was introduced through lectures and in-class exercises over four consecutive class meetings, followed by the second study assignment. The first assignment was not returned to the students until after the second one was submitted so that the grading feedback on the first would not influence the students’ work on the second assignment.

To study how designers work and think, researchers have used a variety of techniques such as protocol analysis, interviews, and direct observation. However, none of these techniques were appropriate for the pilot study, since the 46 students in the two classes would not be working on the problems at a particular time or place. Because students in both courses were already required to maintain journals of their course work, additional journal documentation requirements were added for the assignments and copies of these journal sections were submitted with each homework.

Other researchers have used design journals to capture data about how designers work[232], and using the journals overcomes some of the limitations of other methods. Unlike interviews, data are collected in real time instead of retrospectively, while minimizing the possible effects of altered behavior that can occur with direct observational methods. Data can be categorized using coding techniques, but unlike protocol analysis, the data is collected in situ, minimizing the researcher’s intervention. The major disadvantage of using design journals is that they may not provide a complete record of the students’ design processes. Details may be omitted or inaccurately documented, or students may “backfill” by recording their work in retrospect rather than as it occurs. Backfilling can result in documenting only the successful choices made in the process, omitting the dead ends that helped the student find a better solution.

To help ensure the validity of the journals, several measures were taken. First, students were given in-class instruction in the use of a journal at the beginning of both courses, also noting the professional value of developing the habit of maintaining journals of their work. They were also told to record everything related to the course: lecture notes, reading notes, reflections on conversations with other students, etc. For the second assignment, the use of the Decision Pattern was added for documenting design decisions. Despite these efforts, however, not all journals were usable for this study due to journal quality or because they were not submitted with both assignments. Therefore, journals were selectively sampled for the study, choosing those that appeared complete, consistent with the artifacts submitted, and were without significant or obvious backfill or gaps.

After both assignments were completed, journals were collected and coded using a three part coding scheme. This scheme was designed to provide insight into students’ ability to employ the key expert design strategies: maintaining a system-wide view of the design problem,
opportunistically navigating the design space, and making confident, well-founded design decisions. Each design decision was assigned three codes, with the first two codes characterizing the decision type and perspective or level as shown in Table 2.1.

Table 2.1: Design Decision Coding Matrix

<table>
<thead>
<tr>
<th></th>
<th>Concept</th>
<th>System</th>
<th>Detail</th>
<th>Human</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition (De)</td>
<td>C/De</td>
<td>S/De</td>
<td>D/De</td>
<td>H/De</td>
</tr>
<tr>
<td>Differentiation (Di)</td>
<td>C/Di</td>
<td>S/Di</td>
<td>D/Di</td>
<td>H/Di</td>
</tr>
<tr>
<td>Abstraction (Ab)</td>
<td>C/Ab</td>
<td>S/Ab</td>
<td>D/Ab</td>
<td>H/Ab</td>
</tr>
<tr>
<td>Instantiation (Is)</td>
<td>C/Is</td>
<td>S/Is</td>
<td>D/Is</td>
<td>H/Is</td>
</tr>
<tr>
<td>Composition (Co)</td>
<td>C/Co</td>
<td>S/Co</td>
<td>D/Co</td>
<td>H/Co</td>
</tr>
<tr>
<td>Integration (It)</td>
<td>C/It</td>
<td>S/It</td>
<td>D/It</td>
<td>H/It</td>
</tr>
</tbody>
</table>

The type code delineates six kinds of design decisions that indicate what the designer is doing to the developing design. Decomposition (De) decisions separate a problem into smaller parts with simpler solutions that can then be recombined or Composed (Co) to form a solution to the larger problem. Decisions that generalize a problem design element were denoted as Abstraction (Ab) decisions. Assigning specific details and unique attributes to an element Instantiates (Is) it. Differentiation (Di) is not problem-focused as decomposition is and is used to distinguish different aspects or elements of the problem or system. Likewise, Integration (It) decisions are similar to composition but require a broader view of how the parts of the system are unified into a whole.

The designer’s perspective in making a design decision was assigned one of four levels. Concept design decisions (C) address a problem or sub-problem with preliminary ideas, strategies, and/or approaches. System level design decisions (S) define the needed subsystems, their configuration, and their interfaces. Detail design decisions (D) focus on quantifying specific features required to realize a particular concept. Also considered is the Human perspective (H), that of a user or other stakeholder in the system. This perspective is particularly important as it provides an external view of the proposed solution as it will be situated when complete.

A third coding was performed to characterize the basis or justification used to make a particular design decision. Five codes were used: Requirements or problem statement (R), Principles or best practices (P), Formal or experimental analysis (A), Personal experience (E), and Unknown or not clearly identifiable (U). This three-part coding scheme allowed fine distinctions between students’ design decisions to be identified, for example system-level differentiation based
on design principles versus system decomposition based on personal experience.

To code the journals, the principle investigator and two assistants first carefully read a submitted journal while referring to the associated design artifacts. The journals were then read again to identify decision points. The identified decisions were then discussed among the coding group to reach consensus on these points. Next, each decision was coded independently using the scheme described above. The independent codings were then discussed and reconciled among the coders before being recorded in a spreadsheet.

These three-level codings were then analyzed to identify sequences of three or more design decisions that suggested a trend in the subject’s thought processes and design behavior. For example, a sequence of decomposition decisions following a path from a high-level problem directly to a specific and compact problem-solution was labeled as “linear decomposition.” A decision sequence differentiating and bounding different parts of a problem or subproblem was labeled as “problem-framing.” The most common decision sequence labels identified in this meta-coding analysis are shown in Table 2.2.

<table>
<thead>
<tr>
<th>Novice-type</th>
<th>Expert-type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System View</strong></td>
<td></td>
</tr>
<tr>
<td>Narrow focus</td>
<td>Maintains system view</td>
</tr>
<tr>
<td>Fit problem to solution</td>
<td>Problem framing</td>
</tr>
<tr>
<td>Early concretization</td>
<td>Use of abstraction</td>
</tr>
<tr>
<td><strong>Navigation</strong></td>
<td></td>
</tr>
<tr>
<td>Linear search</td>
<td>Opportunistic search</td>
</tr>
<tr>
<td>Single solution considered</td>
<td>Alternative generation</td>
</tr>
<tr>
<td>Weak relationships</td>
<td>Tolerance for ambiguity</td>
</tr>
<tr>
<td><strong>Decision Making</strong></td>
<td></td>
</tr>
<tr>
<td>Weak justification</td>
<td>Principles/best practices</td>
</tr>
<tr>
<td>Personal experience</td>
<td>Formal analysis</td>
</tr>
<tr>
<td>Use of buzzwords</td>
<td>Confidence</td>
</tr>
</tbody>
</table>

This meta-coding illuminated larger patterns of behavior that could then be compared with existing characterizations of novice and expert design processes, highlighting changes in individual students’ approaches to the second design problem.
2.4.1 Pilot Study Results and Conclusions

There were two goals for this study: (1) Determine if students could (and would) use the $P^3F$ tools to help them work through complex, open-ended design problems; and (2) Identify changes in design behavior that could be associated with students’ use of the framework. The graded assessments of the student submissions were not used as data for this study because they did not provide any additional information or insight related to the study goals.

A total of 38 students in the two classes completed both assignments, 24 in the graduate software engineering class (G:24) and 14 in the undergraduate software design class (U:14). Of these submissions, 29 (G:17, U:12) were determined to be usable for the study based on their completeness, lack of significant gaps and/or backfill, and correlation with the design artifacts submitted. The journals were coded as described and the codings were then analyzed along with the design artifacts from both assignments to identify characteristics of novice and expert design strategies and behaviors. The results are shown in Tables 2.3 and 2.4, which lists the three most frequently identified behaviors in each of the expert strategy areas for novices and expert, respectively. In both tables, the numbers following “G:” and “U:” are the number of instances of an identified behavior for Graduate and Undergraduate students.

<table>
<thead>
<tr>
<th>Table 2.3: Identified Novice Design Behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System View</strong></td>
</tr>
<tr>
<td>Narrow focus</td>
</tr>
<tr>
<td>G:13, U:10</td>
</tr>
<tr>
<td>Fit problem to solution</td>
</tr>
<tr>
<td>Early concretization</td>
</tr>
<tr>
<td><strong>Navigation</strong></td>
</tr>
<tr>
<td>Linear search</td>
</tr>
<tr>
<td>G:9, U:11</td>
</tr>
<tr>
<td>Single solution considered</td>
</tr>
<tr>
<td>Weak relationships</td>
</tr>
<tr>
<td><strong>Decision Making</strong></td>
</tr>
<tr>
<td>Weak justification</td>
</tr>
<tr>
<td>G:15, U:10</td>
</tr>
<tr>
<td>Personal experience</td>
</tr>
<tr>
<td>Use of buzzwords</td>
</tr>
</tbody>
</table>

The data presented in Table 2.3 indicates a general decline in the incidence of novice design behaviors when the students used the $P^3F$ tools to work on the second design problem. The
analysis also suggest that some habits are difficult to break: five of the six students (G:3, U:2) who tried to fit problem 2 to a particular solution showed the same behavior with problem 1. They were also the same five individuals who considered only one solution that was defined and detailed early in their design process, with decisions based only on personal experience. In all five of these cases, the proposed solution used the Linux-Apache-MySQL-PHP (LAMP) model even though both problems explicitly stated that the customer was already using other technologies and did not want to invest in replacements.

Table 2.4: Identified Expert Design Behaviors

<table>
<thead>
<tr>
<th></th>
<th>Problem 1</th>
<th>Problem 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System View</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maintains system view</td>
<td>G:2, U:1</td>
<td>G:13, U:10</td>
</tr>
<tr>
<td>Problem framing</td>
<td>G:5, U:1</td>
<td>G:16, U:10</td>
</tr>
<tr>
<td>Use of abstraction</td>
<td>G:12, U:3</td>
<td>G:14, U:6</td>
</tr>
<tr>
<td><strong>Navigation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opportunistic search</td>
<td>G:1, U:1</td>
<td>G:15, U:11</td>
</tr>
<tr>
<td>Alternative generation</td>
<td>G:5, U:2</td>
<td>G:10, U:8</td>
</tr>
<tr>
<td>Tolerance for ambiguity</td>
<td>G:9, U:1</td>
<td>G:13, U:4</td>
</tr>
<tr>
<td><strong>Decision Making</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal analysis</td>
<td>G:11, U:0</td>
<td>G:10, U:2</td>
</tr>
<tr>
<td>Confidence</td>
<td>G:6, U:7</td>
<td>G:16, U:12</td>
</tr>
</tbody>
</table>

Table 2.4 presents the incidences of expert design behaviors identified in the analysis of the students submissions. This data indicates a general increase in the number of students who demonstrated behaviors associated with expert designers after being introduced to the \( P^3F \). Overall, the most significant increases occurred in the areas of system view maintenance, problem framing, and opportunistic movement through the design and problem space. Additionally, every student that exhibited expert behaviors on the first problem continued to demonstrate those behaviors while working on the second problem.

This study examined the relationships between the changes in observed behaviors, the subjects’ student type (graduate or undergraduate), and their behavior before and after being introduced to the \( P^3F \). Specifically, whether or not there is evidence for a statistically significant change in the observed behaviors that could be a result of the subjects’ introduction to the
For each of the behaviors listed in Tables 2.3 and 2.4 the number of subjects that exhibited that behavior before and after the introduction of the $P^3F$ was counted. During the coding and analysis of the students’ design journals each of the behaviors was independently identifiable, e.g., a single subject could exhibit both linear and opportunistic search behaviors at different times during his or her work on one problem, with both instances recorded. Because the behaviors were independently identified, they can be assumed to be independent for the purpose of this analysis, and examine each behavior separately, before and after the introduction of the $P^3F$.

Viewed from this perspective, there were eighteen independent tests, one for each of the listed behaviors, for each subject before and after the “treatment,” e.g., the introduction of the $P^3F$. For each test, the dependent variable was the indicated behavior which is either present or not present. The before and after tests were correlated for each subject. A widely used statistical tool for determining the significance of changes in a dichotomous variable between two correlated samples is the McNemar symmetry test\[81, 216, 226\]. The McNemar test provides a $\chi^2$ value that is then used to compute the significance level or $p$-value\[179\].

The McNemar statistics were computed using all of the subject data (ignoring the distinction between graduate and undergraduate students) with the results shown in Table 2.5. The null hypothesis for this analysis is that the introduction of the $P^3F$ would have no effect on the subjects’ observed expert and novice behaviors. Based on a 5\% level of significance ($\alpha = 0.05$), the McNemar test statistic is evidence for rejecting the null hypothesis for all but one behavior, formal analysis. This is not surprising since the $P^3F$ does not directly invoke any formal methods for software design analysis and students at the undergraduate and first-year graduate levels usually have had little or no exposure to formal methods.

Tables 2.6 and 2.7 present the same analysis distinguishing between graduate and undergraduate student subject populations. For graduate students, $p < \alpha$ for only 8 of the 18 behaviors, split evenly between the reduction of novice behaviors and amplification of expert behaviors. These results suggest that the $P^3F$ had the most impact on graduate students in the design decisions are grounded and how the system is viewed as the design is developed and refined. The $P^3F$ appears to have had a greater effect on undergraduate students, with 14 of 18 observed behaviors showing significant differences before and after the introduction of the $P^3F$. The four behaviors lacking significant differences include areas commonly difficult for novices to grasp and apply effectively: abstract thinking in the context of ambiguous and ill-structured problems.

Overall, this analysis suggests that the introduction of the $P^3F$ did have a significant effect upon the subjects, reducing the observed incidence of novice behaviors and increasing the incidence of expert behavior. As with most research, this study has limitations. The study subjects were students enrolled in undergraduate and graduate courses, thus extrapolation to other learning environments (i.e., professional development and training) may not be applicable.
Table 2.5: McNemar Test Statistics for All Subjects

<table>
<thead>
<tr>
<th>Behavior</th>
<th>McNemar $\chi^2$</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrow focus</td>
<td>12.250</td>
<td>0.0004</td>
</tr>
<tr>
<td>Fit problem to solution</td>
<td>7.634</td>
<td>0.0067</td>
</tr>
<tr>
<td>Early concretization</td>
<td>7.143</td>
<td>0.0075</td>
</tr>
<tr>
<td>Linear search</td>
<td>4.000</td>
<td>0.0468</td>
</tr>
<tr>
<td>Single solution considered</td>
<td>8.000</td>
<td>0.0047</td>
</tr>
<tr>
<td>Weak relationships</td>
<td>16.000</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>Weak justification</td>
<td>19.000</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>Personal experience</td>
<td>12.250</td>
<td>0.0005</td>
</tr>
<tr>
<td>Use of buzzwords</td>
<td>13.000</td>
<td>0.0003</td>
</tr>
<tr>
<td>Maintains system view</td>
<td>20.000</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>Problem framing</td>
<td>20.000</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>Use of abstraction</td>
<td>5.000</td>
<td>0.0253</td>
</tr>
<tr>
<td>Opportunistic search</td>
<td>24.000</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>Alternative generation</td>
<td>8.333</td>
<td>0.0039</td>
</tr>
<tr>
<td>Tolerance for ambiguity</td>
<td>5.444</td>
<td>0.0196</td>
</tr>
<tr>
<td>Principles/best practices</td>
<td>11.000</td>
<td>0.0009</td>
</tr>
<tr>
<td>Formal analysis</td>
<td>0.333</td>
<td>0.5637</td>
</tr>
<tr>
<td>Confidence</td>
<td>15.000</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

The study was also limited in scope and duration, so the results should not be taken to reflect long-term learning. Students solved only one problem using the $P^3F$, so these results cannot be generalized to other types of design problems. However, this was in line with the goals of the study: to determine if students could apply the tools to a problem, and if that use would effect any change in their design behaviors.

The data used in this study also presents difficulties. Design journal data, particularly when compiled in a graded academic setting, are potentially inaccurate, incomplete, or biased, and represent self-reported data. Efforts were taken to minimize these effects by providing instruction in document design work and providing a specific tool (the Decision Pattern) for the second assignment. Including the journal in the assignment grading provided an incentive for students to be complete and accurate in their reporting, and the journals were extensively reviewed in conjunction with the submitted design artifacts to validate their completeness and accuracy.

Finally, this study did not consider the quality of the design artifacts. In an academic setting, students tend to be conditioned to producing the “correct” or expected answers[109]. At this stage of the research, the goal was to identify changes in behavior indicating students’ use of the framework. Identifying and quantifying changes in the quality of the design artifacts will be
addressed in future work.

The results of this pilot study suggest that after being introduced to the $P^3F$, students’ tendencies to engage in behaviors characteristic of novices declined, and their tendency to exhibit expert behaviors increased. This suggests that this sample of students were able to understand and use the learning tools in the $P^3F$, and that when they did, their work took on aspects of expert designers’ strategies. However, this conclusion must be tempered with the knowledge that these students were doing homework using a prescribed tool set that was designed to produce these effects.

### 2.5 Adapting Expert Strategies to Help Novice Designers

At the beginning of this chapter, I noted that the literature of learning to design software systems is minimal, but that there have been informative studies of experts and novices in a variety of design-intensive disciplines. In the preceding sections, I have reviewed these studies from three perspectives: how design problems are viewed, how the evolving problem/solution space is managed and navigated, and how design decisions are made. These reviews provide us with answers to the first two research questions posed at the start of this chapter, and this
Table 2.7: McNemar Test Statistics for Undergraduate Students

<table>
<thead>
<tr>
<th>Behavior</th>
<th>McNemar $\chi^2$</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Narrow focus</td>
<td>9.000</td>
<td>0.0027</td>
</tr>
<tr>
<td>Fit problem to solution</td>
<td>6.000</td>
<td>0.0143</td>
</tr>
<tr>
<td>Early concretization</td>
<td>8.000</td>
<td>0.0047</td>
</tr>
<tr>
<td>Linear search</td>
<td>6.000</td>
<td>0.0143</td>
</tr>
<tr>
<td>Single solution considered</td>
<td>7.000</td>
<td>0.0082</td>
</tr>
<tr>
<td>Weak relationships</td>
<td>9.000</td>
<td>0.0027</td>
</tr>
<tr>
<td>Weak justification</td>
<td>6.000</td>
<td>0.0143</td>
</tr>
<tr>
<td>Personal experience</td>
<td>8.000</td>
<td>0.0047</td>
</tr>
<tr>
<td>Use of buzzwords</td>
<td>9.000</td>
<td>0.0027</td>
</tr>
<tr>
<td>Maintains system view</td>
<td>9.000</td>
<td>0.0027</td>
</tr>
<tr>
<td>Problem framing</td>
<td>9.000</td>
<td>0.0027</td>
</tr>
<tr>
<td>Use of abstraction</td>
<td>3.000</td>
<td>0.0833</td>
</tr>
<tr>
<td>Opportunistic search</td>
<td>10.000</td>
<td>0.0016</td>
</tr>
<tr>
<td>Alternative generation</td>
<td>3.571</td>
<td>0.0588</td>
</tr>
<tr>
<td>Tolerance for ambiguity</td>
<td>3.000</td>
<td>0.0833</td>
</tr>
<tr>
<td>Principles/best practices</td>
<td>8.000</td>
<td>0.0047</td>
</tr>
<tr>
<td>Formal analysis</td>
<td>2.000</td>
<td>0.1573</td>
</tr>
<tr>
<td>Confidence</td>
<td>5.000</td>
<td>0.0253</td>
</tr>
</tbody>
</table>

section will develop an answer to the third: “Which of the identified expert strategies would directly address the critical weaknesses of novice design approaches?”

This section begins with a summary and comparison of expert and novice design behaviors as identified in previous studies of designers, identifying the expert strategies that could provide the most immediate benefit for novice software designers. This is followed with a brief discussion of the insights and lessons learned from our pilot study as viewed through the lens of the literature review. This section concludes the chapter by summarizing advice and suggestions for helping students in design-intensive disciplines develop more mature design skills, laying the foundation for the Principles, Patterns, and Process Framework elaborated in the next chapter.

2.5.1 Summary & Comparison of Expert and Novice Design Behaviors

When the behaviors and strategies of novices and experts are compared, diametrically opposed characteristics are generally not found. What is indicated is that novices lack the tools and experience to deal with problems that are complex, ill-formed, and do not lend themselves to the deductive, analytical problem-solving techniques they have learned to rely on in their academic experience. This lack is evident in all three strategic areas. Since professional experience cannot be generated out of thin air, comparisons of experts and novices must focus on the thinking
tools experts employ to help them effectively solve difficult design problems.

As the literature shows, novice decision-making strategies tend to be linear, deductive, and analytic, a result of the way they are trained to solve problems in academic settings. Novices seek concrete and objective literal data upon which to base their decisions. While experts also employ deduction and objective data in their decision making, they also apply naturalistic and abductive strategies to decision-making. Experts also often use subjective information to make value-based decisions to balance opposing forces in a design situation rather than a raw optimization strategy. To help novices behave more like experts, they need tools that appeal to their analytic background while enabling them to think more subjectively.

A key expert strategy for managing and navigating the evolving design problem/solution space is the ability to arbitrarily frame chunks of the design in order to control the scope of what one is working on at a given point in time. Augmenting this strategy is the use of abstractions and metaphors to label these chunks in mentally manageable ways. This enables the expert designer to move about in the design space opportunistically and to reflect back on the current, future, and past states of the design and the path to the current situation in the design process. The abstractions also carry relational information that helps the designer hold everything together. Novices, on the other hand, tend to decompose problems using a trial and error process that focuses on only one part of the problem at a time, leading to difficulties when it is time to integrate solutions to subproblems. Tools that would help novices frame and chunk the problem/solution space and describe relationships between those chunks could benefit novices’ design performance.

Because novice designers tend to have a problem-centric view, they often fail to understand and appreciate the larger-scale consequences of their choices as they develop a design solution. Expert designers do understand these consequences, and strive to maintain a big picture view of the design situation that includes the overall environment and application space of the system. Related human activities, stakeholder perspectives and needs, and interactions with other software systems can be critical to the success or failure of a system. Novices need guidance and persuasion to help them understand the importance of these non-software concerns, and to recognize and incorporate these relationships and interactions into their design thinking.

### 2.5.2 Insights from the Pilot Study

The insights gained from the pilot study come from the study result as well as from anecdotal evidence drawn from end-of-course student evaluations and students’ design journals. Both perspectives offer valuable information about how students reacted to their exposure to the $P^3\mathcal{F}$, and helped to guide the refinement of the framework in preparation for the current work.

The analysis of the pilot study data showed significant increases in expert-like behaviors
among the subjects as well as decreases in design behaviors associated with novices. In particular, the results suggest that students benefited most in the behaviors related to how they viewed and navigated the design space, and in their confidence in their design decisions. The subjects showed dramatic improvement in framing and maintaining a big-picture view of the problem and solution. This is an aspect of expert behavior that I see as fundamental: If the designer cannot maintain a consistent and comprehensive view of the developing system, most of the other expert strategies have little grounding or value.

The Decision Pattern, an integral part of the P3F and a required part of the pilot study subjects’ design documentation for the second problem, provided students with a way to bring some structure to the design problem, and the course content in general, while allowing themselves to think more openly than most of their other academic experiences would allow. With familiarity and practice, the critical thinking process embodied in the pattern became integrated into their natural thought processes. One student wrote in his design journal:

*The decision pattern is just a tool of the critical thinking process, but this pattern and this course in general has helped me to enhance my critical thinking by analyzing the reasons why I do or do not do things — be that software design or just things in general.*

The need to effectively communicate design knowledge with others was a common theme in the students’ journals. Many noted that standard design notations (such as the various types of UML diagrams) communicate the design itself, but do not easily incorporate the assumptions, forces, and sequences of decisions that result in those notational artifacts. Several students also noted that in order to construct a solid understanding of the design artifacts and their relationships, they needed this tacit information about the history and evolution of the artifacts. Going a step further, several wrote that understanding how the parts of a system are related is vital to understanding the system as a whole.

Another insight came from a group of students in the class who consistently applied the P3F principles in their justifications for solutions to the assigned design problems. These students performed significantly better than their peers who did not apply the principles as consistently. Specifically, these students correctly identified more necessary design elements in the given problems, which were deliberately constructed to be outside their previous experience. These students were also the only ones to discover latent dilemmas built into the problems that required more information than was given. These assessments seem to indicate that these principles may stimulate and help structure critical thinking skills in a software design context.
2.5.3 Suggestions for Improving Design Skills

While many of the studies of novice and expert designers report only the descriptive results of their research, there are some that offer suggestions for helping novice designers overcome their weaknesses and strengthen expert-like behaviors. Other authors provide advice based on professional practice in design-related disciplines or from their experiences teaching novice designers. Examining these suggestions and advice can provide further guidance and grounding for the Principles, Patterns, and Process Framework.

I begin with advice directed towards helping novices develop the ability to view design problems in ways similar to those used by expert designers. Dym suggest that novices need support for seeing and understanding the relationships and interactions between components in large, complex systems[89]. Cardella, et. al., note the importance of motivating students to consider the “big picture” that includes business, ethical, and socio-cultural implications of proposed solutions[45]. One way to support this big picture view is to situate design problems in a particular context that includes dimensions such as economic, environmental, social, and user-centric issues across diverse cultures. Students need tools that support the identification and elaboration of contextual information that is not directly part of the system’s implementation but is critical for the system’s operation[45, 151].

Tesar suggests that how a problem is viewed affects how it is solved, and that novice designers often already have predispositions to particular design perspectives, and to overcome these biases they should explicitly and actively look at the problem from different points of view[241]. Hult, Irestig, and Lundberg propose a generic perspective framework providing eight dimensions (users, artifacts, context, activities, central relations, communication, perspective-implied use qualities, and perspective-specific vocabulary) that can be used to construct and describe specific perspectives on a design problem[135]. Their research suggests that such a framework can be a constructive way to explore and understand the design task, particularly in the early stages of the design process. Frameworks such as this must also accommodate multiple strategies and heuristics, and should motivate students to manipulate how they view, scope, and frame design problems[33, 89, 234].

Abstractions and metaphors are important expert tools for managing the evolving design problem/solution space. They incorporate structural, behavioral, and relational information in compact mental models that can be expanded and examined in detail when necessary. Novice designers, on the other hand, have a preference for the concrete and tangible, and need some form of support to help them learn to think abstractly[37, 44]. Guindon and Curtis suggest that collections of abstract design schemas are a promising tool for supporting novice designers. Indeed, exposure to design patterns has been a part of standardized computing curricula for over a decade[4, 2], but the patterns commonly used (i.e., the “Gang of Four” Design Patterns[111])
are often tied very closely to program code, making them little more than templates instead of abstract representations[215].

To help them think more abstractly, students need more patterns and metaphors that can abstractly represent larger elements and subsystems within large, complex software systems[92]. Through abstract descriptions of relationships and behaviors, these schemas or patterns could also help the novice designer maintain a situated vision of the evolving solution as it relates to its complex context[151]. By compactly representing and organizing design elements, they can support the designer’s movement through the design space, allowing the them to think both convergently and divergently as they explore the evolving problem and solution[89, 45].

Thinking abstractly and maintaining a multi-perspective view of the design problem and solution certainly have the potential to improve the performance of novice software designers. However, without a reliable process or framework to guide the use of these tools, novices will not be able to effectively apply them to complex design problems. One perspective on the process of design characterizes it as an inquiry or question driven learning and exploration process[90, 80]. Asking questions is a fundamental aspect of designer’s cognitive activities, and is intimately related to their problem-solving, creativity, decision making, and learning processes[99]. Building on this view, Ahmed and Wallace note that “supporting novice designers by simply supplying knowledge may not be enough, they also require support in identifying what they need to know.”[8, 7]

A framework or process for novice designers must also be flexible to accommodate different strategies for a variety of design-related activities such as knowledge acquisition, abstract modeling and simulation, alternative generation, evaluation, and concretization[33]. Rules, principles, and other kinds of consistent reference points should also be incorporated to provide novices with guidance for making various design decisions[176, 53]. Most of all, a design framework intended for novice software designers must be easily accessible to them[191].

Reflection, in the sense of thinking about something again, is a key aspect of expert designers’ cognitive processes[5]. Schön describes two types of reflection: reflection-in-action and reflection-on-action[221]. Reflection-in-action is the ability of experts to “think what they are doing while they are doing it,” the process of immediately thinking about unexpected events in new ways. This allows the designer to construct an informed and expeditious response to the event. Reflection-on-action, on the other hand, occurs after the fact, and is the process of thinking about why various actions were taken and decisions made, allowing the designer to critically examine past actions and experiences in order to identify potential conflicts and learn from mistakes[221, 222]. The critical thinking and learning aspects of reflection-on-action, in particular, are valuable cognitive skills that novices need help in developing, and should also be incorporated into a process or framework supporting novice designing[140, 127].

In conclusion, I have reviewed prior research studies of expert and novice designers to identify
characteristics of their design behaviors and thinking processes. Critical differences between experts and novices have been highlighted as areas than can be addressed in a framework designed to help novices adopt and nurture expert-like design behaviors. The $P^3F$ was built upon the foundation of these key differences, insights from the pilot study, and pedagogical suggestions. The next chapter will focus these differences, insights, and suggestions into a set of basic requirements to drive the design of the Principles, Patterns, and Process Framework.
Chapter 3

The Principles, Patterns, and Process Framework

The majority of graduating computer science students cannot effectively design a complex software system, with the most common problems being the inability to see the system as a whole, not understanding what kind of information must be present in the delivered system as well as in the system design, and weak communication skills[93]. Effective software system design education must maximize the skills of students, providing all of them with sound foundational principles and practices of design and enabling the exceptional students to be nurtured and developed into the next generation of great software designers[34, 132]. As a learning tool, the Principles, Patterns, and Process Framework (P³F) incorporates the effective strategies of expert designers in a principle-driven, pattern-based framework that includes a generalized iterative design process. As students work within the framework, they use these strategies without the cognitive load of having to “learn” them, and repetition of the underlying decision-making patterns reinforces these strategies and practices.

Software systems are complex and dynamic networks of components that manipulate and exchange information in many ways. Over time, the structure and behavior of a software systems will change as the executable code is modified and extended and as users adapt the functionality to serve new purposes. Software systems are designed to work in a particular computational, physical, and social environment, but as they are used, the systems affect this environment in often unpredictable ways[150, 180]. Architecture has long been a metaphor and model for the design of large software systems because architectural design is also concerned with complex, dynamic systems: buildings, their environment, the people that live, work, and interact with them, and how the system evolves over time are all important considerations for the architect[68]. One architect who has influenced software system design for over forty years is Christopher Alexander, first introduced to the discipline of software design and engineering in 1968 by Peter
Naur:

...software designers are in a similar position to architects and civil engineers, particularly those concerned with the design of large heterogeneous constructions, such as towns and industrial plants. It therefore seems natural that we should turn to these subjects for ideas about how to attack the design problem. As one single example of such a source of ideas I would like to mention: Christopher Alexander: “Notes on the Synthesis of Form.”[190].

Alexander’s most influential work, with respect to software design, has been in the area of design patterns and pattern languages[17, 12]. The use of patterns and pattern languages to capture best practices within particular domains is widespread[67, 95, 111, 206] and has even been integrated into enterprise-class software system development tools such as IBM’s Rational collection of products. In 1996, Alexander presented a new view of architectural theory to the computing community, urging researchers in computer science and software design to “become responsible for the form and structure of the built environment,” in particular, the ways in which computing directly and indirectly affects the world we live in[13]. Thus far, this abstract and theoretical approach to “living architecture” has attracted only modest attention from researchers and practitioners in this discipline[64, 65, 106, 107, 207, 236].

Alexander’s work deals with complex systems that are under the constant pressure of change. In the four volumes of The Nature of Order he relates the dynamic complexity of urban architecture with living systems, building architectural models that incorporate this dynamic structure and behavior. Software systems are large, complex systems that function in a dynamic environment in much the same manner as physical buildings within a community are dynamic systems when considered in their context. As an approach to designing and constructing “living systems,” Alexander’s work provides a starting point for the development of the Principles, Patterns, and Process Framework.

The goal of this chapter is to answer the fourth of our research questions: How can these expert strategies be incorporated into a novice-oriented knowledge-based learning and application framework? We begin in Section 3.1 by distilling what we have learned from the review of prior and related work into a set of core requirements for our framework. To provide a context for the $\mathcal{P}^3\mathcal{F}$, Section 3.2 briefly discusses software design principles that are widely regarded as foundational to good software system design. The Principles, Patterns, and Process Framework is presented in Section 3.3.
3.1 Characteristics of a Framework to Support Novice Software Designers

The objective of this section is to identify some basic characteristics of a design support framework and what it should and should not provide for novice software designers. In the previous chapter we reviewed prior work in order to characterize common novice and expert design behaviors as well as collecting suggestions and advice regarding ways to cultivate expert design skills. As part of this review we contrasted expert and novice behaviors and strategies to determine a set of key skills that could help novices improve their design performance. We will begin this section by summarizing these skills followed by a discussion of how they can be effectively packaged and communicated to software design students.

The key set of skills in the typical expert designer’s toolbox are those related to manipulating how the design problem and solution are viewed. Experts maintain a big picture view that incorporates the full context of the problem and the environment where the solution will exist. Novices, on the other hand, often become focused on the problem only as a description of a concrete implementation that they attempt to fit into the structure of other familiar problems. Experts control their views and perspectives in order to understand the problem and solution, while novices are controlled by their superficial understanding of the problem as it is given.

This narrow focus is at the center of other weaknesses novice designers exhibit. For example, novices have difficulty seeing and understanding the relationships and interactions between parts of large, complex software systems. This problem extends to understanding the relationships between the system and other systems it interacts with, including the people and organizations that use or depend on the system. Their implementation-driven view makes abstractions unnecessary other than those that are part of the programming language(s) used to code the system.

The inability to look at the problem from different perspectives and different levels of abstraction can also affect how the novice navigates the design space and generates alternative solutions. Because the view is usually tightly focused on a particular problem or subproblem, relationships between this problem and other parts of the system are ignored. This is also evidenced by the tendency of novices to use a depth-first approach to move quickly from conceptual problem statements or requirements to the implementation of specific details with little regard for how their choices may affect other parts of the system.

The novice’s restricted view of the design problem and solution also manifests itself in the way design decisions are made. Academic training provides them with a variety of “plug-and-crank” problem-solving techniques that focus attention on distinguishing what is known from what is unknown. The known elements of the problem can then be manipulated with formulas or other structured procedures to generate the unknown elements. These techniques leverage and reinforce
the narrow, literal view of the design problem. Faced with the uncertainty and ambiguity that is typical of software system design problems, these linear, deductive problem-solving tools fail the novice.

Because novice designers prefer to view the problem and solution in a literal and concrete fashion as much as possible, they are resistant to revisiting design decisions they have already made, particularly when such a review or reconsideration could entail changing or abandoning previous decisions. Their design decisions are usually made in relative isolation from each other, and little consideration is given to the sequences of decisions that have been made up to that point. Likewise, there is a lack of appreciation for the future consequences of design decisions.

From this discussion we can see that some kind of support for constructing, maintaining, and manipulating a comprehensive and situated view of a design problem and evolving solution is a critical need for novice software designers. Novices also need a toolkit of “general-purpose” abstractions they can relate to and use to help them manage a big picture design view. These abstractions will serve three purposes with respect to their view of the design problem/solution: they will help minimize cognitive load, they will support thinking about the problem and solution in less literal ways, and they will provide a way to defer decisions by temporarily compartmentalizing uncertainty and ambiguity.

We also learned from our review that students like to have formal processes or procedures to use for solving problems, but as we have seen, the analytical procedures they learn to use do not work well with ill-formed problems. A flexible and adaptable “design thinking” process for guiding design decision-making would provide an alternative to these formal procedures. This process could explicitly incorporate steps for looking at the big picture of the system, the use of rules, principles, and/or practices for examining the design and generating solution alternatives, selecting and evaluating candidate alternatives, and reflecting on what has been accomplished.

### 3.2 Foundational Software Design Principles

Design decisions do not “just happen” without some motivation and justification. Expert designers rely on fundamental principles to identify when a decision needs to be made, and to provide the rationale for the choice that was made[75]. In addition to incorporating the strategies of expert designers, the $P^3F$ must also provide a principled foundation to support novice design decision-making. We consider several collections of basic software system design principles with two related goals: 1) to identify a set of commonly understood foundational principles for quality software system designs; and 2) to form a basis for the refined set of design principles that are presented as a cornerstone of the $P^3F$.

Describing the basis and content of an undergraduate software engineering course, Hoffman[132] identifies four fundamental principles of software design, summarized in Table 3.1. We can
see that even in this small set of principles there are intertwining relationships between the principles. For example, the *Information Hiding* principle requires that important design and implementation decisions be hidden behind a stable interface so that the implementation may be changed without altering the external behavior of the software module. In order to accomplish this, however, the details of the design and implementation are *Abstracted* out of the interface that the module presents to its users. Because the implementation is hidden behind an abstract interface, proper *Documentation* is necessary in order to communicate the behavior and functionality of the module to potential users.

Table 3.1: Fundamental Software Design Principles[132]

<table>
<thead>
<tr>
<th>Principle Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separation of concerns</td>
<td>Complexity management through decomposing problems into smaller, independent problems (from [86, 182]).</td>
</tr>
<tr>
<td>Information hiding</td>
<td>Hiding design and implementation behind stable interfaces (from [198, 208]).</td>
</tr>
<tr>
<td>Abstraction</td>
<td>Removing unnecessary features to simplify a problem or factoring out common information to develop a more general solution that can be specialized as needed (from [134, 197]).</td>
</tr>
<tr>
<td>Documentation</td>
<td>Enabling and mediating communication throughout the life of the software system (from [132, 251]).</td>
</tr>
</tbody>
</table>

Witt, Baker, and Merritt distill extensive industrial software development experience into four fundamental axioms of software system design (Table 3.2) and five fundamental design principles (Table 3.3). These axioms and principles have much in common with those presented by Hoffman, further corroborating their importance and efficacy to software system designers. In particular, the themes of modularity, abstraction, and information hiding are especially conspicuous in Witt, et. al.’s descriptions, indicating the maturity and widespread acceptance of these principles in practice[253]. The emphasis on modular systems with reusable components that are described by abstract interfaces was a significant factor in the development of Object-Oriented programming languages and related analysis and design concepts.

As Object-Oriented software development matured, basic design principles were restated from this analysis, design, and implementation perspective. Martin [174] identified ten such principles, shown in Table 3.4. Note that modularity and information hiding are not explicitly stated in these principles, but both concepts are foundational to most of them. The idea of an abstract, extensible interface that hides the concrete realization of a particular functionality,
Table 3.2: Fundamental Axioms of Software Architecture Design[253]

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separation of Concerns</td>
<td>A complex problem can best be solved by initially devising an intermediate solution expressed in terms of simpler independent problems.</td>
</tr>
<tr>
<td>Comprehension</td>
<td>The mind cannot easily manipulate more than about seven items at a time.</td>
</tr>
<tr>
<td>Translation</td>
<td>Design correctness is unaffected by movement between equivalent contexts.</td>
</tr>
<tr>
<td>Transformation</td>
<td>Design correctness is unaffected by replacement with equivalent components.</td>
</tr>
</tbody>
</table>

a hallmark of object-oriented systems, is implicit in all of Martin’s Object-Oriented Design Principles.

Software engineering texts provide students with many collections of design and implementation principles, but these collections are domain-specific, focusing on software system domains such as web-based applications, object-oriented components, database systems, etc.[11, 123, 201, 204, 220]. Furthermore, these principles are presented descriptively, as if they are a result of good design rather than as tools a designer uses to guide and inform the design process. In this form, these principles offer little guidance towards defining and designing the a solution to a problem, a critical aspect of the process of creating quality designs[87, 126, 187]. Yet these are the design principles that are introduced to novice software designers. It is important that they understand these principles and be able to recognize their presence, they also need fundamental principles that will help guide and inform their design decisions.

A resolution for this problem is the development of a more general and fundamental set of design principles for software system design. These principles must also be stated prescriptively so that they can be used by designers to guide and justify design decisions. The risk of generalized design principles is that the process of generalization may eliminate the attributes and behaviors the principles constrain, leaving the principles too general to be useful. The principles are a key part of the $\mathcal{P}_3F$, and strike a balance between generality and specificity. Additionally, they are structured and stated to be useful in a variety of design decision-making situations, and can be used to derive and justify existing domain-specific design principles[255].

44
Table 3.3: Fundamental Principles of Software Architecture Design[253]

<table>
<thead>
<tr>
<th>Principle Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular designs</td>
<td>Modularity can be achieved by dividing large aggregates of components into units having loose inter-unit coupling and high internal cohesion, by abstracting each unit's behavior so that its collective purpose can be known, by recording each unit's interface so that it can be employed, and by hiding its design so that it can be changed.</td>
</tr>
<tr>
<td>Portable designs</td>
<td>Portability can be achieved by employing abstract context interfaces.</td>
</tr>
<tr>
<td>Malleable designs</td>
<td>Malleability can be achieved with designs that model an end-user view of the external environment.</td>
</tr>
<tr>
<td>Intellectual control</td>
<td>Intellectual control can be achieved by recording designs (after developing a design strategy) as hierarchies of increasingly detailed abstractions.</td>
</tr>
<tr>
<td>Conceptual integrity</td>
<td>Conceptual integrity can be achieved by uniform application of a limited number of design forms.</td>
</tr>
</tbody>
</table>

3.3 The Principles, Patterns, & Process Framework

A study of software design patterns led to their inspiration in the work of Christopher Alexander[17]. Alexander’s career work has centered on how untrained people create buildings and communities of lasting value[12, 13]. Two important suggestions arose from this study. First, buildings and communities are complex and dynamic systems made up of many heterogeneous elements that must work together. Software systems are also complex, dynamic systems with many different elements (data storage systems, user interfaces, communication systems, etc.) that must also work together efficiently if the system is to function properly. Insights into creating sustainable buildings and communities should also be relevant to the design of complex software systems. The second suggestion resulting from the study of Alexander’s work is that novice, untrained individuals and groups can design and build structures of lasting value and function. This suggests that novice software designers could benefit from understanding how untrained people create buildings of value.

The Principles, Patterns, and Process Framework is made up of four related elements: a set of fundamental design principles, pattern structures that help “chunk” information into manageable and relatable pieces, a process for applying the design principles and navigating through the developing design, and a template for making informed and confident design decisions. Together these elements provide a simple and extensible set of tools that incorporate the expert design strategies noted above in a structure and format that can be used by novice software designers.

Pattern structures are foundational to the $\mathcal{P}^3\mathcal{F}$ because they express both rules for creating
Table 3.4: Principles of Object-Oriented Design[174]

<table>
<thead>
<tr>
<th>Principle Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Closed (OCP)</td>
<td>A module should be open for extension but closed for modification.</td>
</tr>
<tr>
<td>Liskov Substitution (LSP)</td>
<td>Subclasses should be substitutable for their base classes.</td>
</tr>
<tr>
<td>Dependency Inversion (DIP)</td>
<td>Depend upon Abstractions. Do not depend upon concretions.</td>
</tr>
<tr>
<td>Interface Segregation (ISP)</td>
<td>Many client specific interfaces are better than one general purpose interface.</td>
</tr>
<tr>
<td>Release Reuse Equivalency (REP)</td>
<td>The granule of reuse is the granule of release.</td>
</tr>
<tr>
<td>Common Closure (CCP)</td>
<td>Classes that change together, belong together.</td>
</tr>
<tr>
<td>Common Reuse (CRP)</td>
<td>Classes that are not reused together should not be grouped together.</td>
</tr>
<tr>
<td>Acyclic Dependencies (ADP)</td>
<td>The dependencies between packages must not form cycles.</td>
</tr>
<tr>
<td>Stable Dependencies (SDP)</td>
<td>Depend in the direction of stability.</td>
</tr>
<tr>
<td>Stable Abstractions (SAP)</td>
<td>Stable packages should be abstract packages.</td>
</tr>
</tbody>
</table>

structures and the relationships between structures in space and time[63]. They also provide a means for abstractly chunking information into manageable units referenced by a metaphoric name within a consistent context[185]. Software design students should be familiar with design patterns as they are a recommended part of the undergraduate curriculum as well as commonly used in professional practice[4].

The $P^3F$ uses an adaptation of a common design pattern format based on Alexander’s pattern structure to express and document the fifteen fundamental design principles, discussed later in this section.[255]. Use of the pattern structure provides contextual and relational information that helps explain how, when, where, and why each principle is applicable. In addition to providing a basis for design decision-making, these principles also provide an abstract structure for the developing system to help facilitate navigation through the design space through the use of the pattern structure.

The process element of the $P^3F$ is derived from Alexander’s *Fundamental Differentiating Process*[16, pp.215-217]. This process provides a template for identifying and iterating over generative design decision sequences that allow a system’s structure, behavior, and interaction with its environment to unfold as a coherent whole that smoothly integrates with its environment. With each new iteration, the student is directed to refresh the view of the system as a whole, incorporating the changes made in the previous iteration. The student is also directed to evaluate the current system using the design principles as guide to identify the strengths and weaknesses of the system, identifying the part of the system that need the most immediate attention.
The final element of the $\mathcal{P}^3\mathcal{F}$ is a template for making and recording design decisions that we call the Decision Pattern[254]. This template, which is also presented in a pattern format, abstracts and structures the key elements necessary to make a critically-informed decision. Requiring students to use a strict format for documenting their decisions can provide insights into how the student arrived at a particular solution[247]. This is similar to showing each step in a mathematical proof or derivation: a step represents the application of some operation or transformation intended to move the problem closer to a successful resolution. The template also provides the student with a tool for reflecting on a sequence of decisions leading up to a particular point in the design of a software system.

Figure 3.1 graphically illustrates the relationships between the elements of the $\mathcal{P}^3\mathcal{F}$. The fundamental Principles, along with their Pattern presentations, form the foundation of the framework. The Process builds on this foundation, integrating the $\mathcal{P}^3\mathcal{F}$ into a three-point support structure for design thinking. The Process guides the application of the Principles to identify the characteristics of entities within the design space and their relationships to each other. At the same time, the Process leverages the Pattern structure to illuminate the context of the design problem at hand and the larger and smaller scale structure of the problem and evolving solution. The Decision Pattern is nested within the Principles, Patterns, and Process as the realization of a design decision made within a defined context, helping the user to understand how and why that decision was made and the consequences that may result from it.

![Figure 3.1: Graphical view of $\mathcal{P}^3\mathcal{F}$ relationships](image-url)
The remainder of this section describes the $P^3F$ in greater detail, beginning with the core principles of the framework. This is followed by the development of the pattern structure used to present these principles to the novice designer. I then define the “design thinking” process developed to provide a means for the novice to imitate the way expert designers think about complex problems. The Decision Pattern, an abstract template providing an alternative view of the process and a way for the novice to encapsulate design decisions, is then defined. This section concludes with an example of using the framework.

3.3.1 Principles for Relationships and Values

The principles that form the foundation of the $P^3F$ are intentionally abstract. This allows them to be applied to a wide variety of situations that may arise in the development of a software system. These principles capture the structure and organization of robust, sustainable complex systems and fundamental and generative relationships between elements in a system. Software systems are complex, dynamic systems with many different elements (such as data storage systems, user interfaces, and communication systems) that must also work together efficiently within the context of a human organization if the system is to function properly.

Software engineering texts list many “principles of software engineering” such as Modularity, Information Hiding, Open-Closed, etc., associated with specific topics in the text. While these principles are certainly valuable within the scope of the associated topic, they are specific to that topic. For example, Pressman devotes an entire chapter to principles of software engineering practice, part of another chapter to principles of object-oriented component design, and part of another chapter to design principles for web-based applications[204]. The $P^3F$ resolves these limited-scope principles through a set of more general and abstract principles for software system design. This abstractness is not without significant risk, since the degree of generality of a principle is directly proportional to the set of attributes or behaviors that the principle constrains. These principles strike a balance between generality and specificity, and are structured such that they may be applied to design situations at different levels of detail or abstraction.

Alexander identifies fifteen properties of living, sustainable systems, whether natural or human-built[15, pp. 145-235], [14]. In the second volume of The Nature of Order, these properties are restated as generative functions Alexander calls structure-preserving transformations[16, pp. 66-76]. Taken together, these dual definitions of property and transformation describe the attributes, behaviors, and relationships between elements of complex, dynamic, and sustainable systems. Since a principle is a rule or standard that determines the intrinsic nature or behavior of something[82], we can interpret these property/transformation definitions as principles that ground and sustain complex systems[255].

To understand the properties, transformations, and ultimately the principles that we have
derived from them, we must have an understanding of the concept of a center. From one perspective, a center is any thing or entity that we are interested in or that attracts our interest to itself. A center is complete in itself, but must also be a part of something else. In the broadest sense, centers span the range from the entire universe (if we could observe it from the outside) through galaxies, star systems, and planets to molecules, atoms, electrons, protons, and the universe of other sub-atomic particles that make up our physical experience.

In the realm of software systems, we have a similar structure of centers. A single program is a center within an operating system, which is itself a system of centers (programs, services, user interface elements, and other system components) that provide the interface or boundary between the hardware and the user. Within a program, we have various code entities (classes, objects, structs, etc.) that are made up of various statements in a programming language. These statements are constructed from expressions and operators, which are made of particular sequences of characters that are human-readable symbols representing strings of 1’s and 0’s. Each level of detail and abstraction generates centers at that level, centers that encapsulate and hide lower-level details and that become building blocks for higher-level centers.

Alexander, speaking from the physical world of architecture, describes a center as “a distinct set of points in space, which, because of its internal coherence, and because of its relation to its context, exhibits centeredness, forms a local zone of relative centeredness with respect to other parts of space.”[15, p. 84] This sounds like a cyclic definition, and it is, to an extent. An entity within a system is a center because it is a focal point of attention, information transfer, action, or other kind of attraction. Other than at the atomic level (which is dependent upon our current perspective on the system), centers are composed of other centers. For example, if we are interested in single-celled organisms that consume a particular substance in their environment, the organisms and pieces of food would be centers. However, if we are interested in how that organism digests and metabolizes the food, we must look inside the cell to the various organelles that carry out the chemical processes. These entities and their processes are now our centers of interest.

Alexander’s fifteen fundamental properties of living systems describe systems of centers, the relationships between those centers, as well as the location and interaction of the system with its context. Like the definition of centers, these properties have a recursive nature. As described by Alexander, these properties are generally visual and geometric, and they are intertwined in many ways. The strong presence of any one of these properties will require the presence of others, and will strengthen or intensify other properties within a system. These properties are summarized below:

1. **Levels of Scale** - Centers tend to have a range of sizes that exist at well-defined levels with definite jumps between them. The differences in scale within a system are not merely
coincidental, but are intentional means of strengthening larger centers with proportionally smaller ones that add detail and intensity. The best proportions are between 2:1 and 4:1. Changes much less than 2:1 are difficult to distinguish. Likewise, jumps of 10:1 or more break cognitive continuity because there is too much difference to maintain a connection between the larger and smaller centers[15, pp. 145-150].

2. **Strong Centers** - While systems are made of centers, strong centers are those parts of the system that are vital and central to its existence. Strong centers are made strong by the field of centers that surround them and by the importance placed on them by their place within the system[15, pp. 151-157].

3. **Boundaries** - Boundaries serve several purposes in complex systems. They focus attention on a particular center, strengthening it within its locale. Boundaries also unite the center with its environment. Most importantly, boundaries are themselves a system of centers that surround, enclose, separate, and connect a center within a field or system of centers[15, pp. 158-164].

4. **Alternating Repetition** - Alternating repetition represents the static and dynamic rhythm of a system of centers. Repetition helps to strengthen centers, and alternation between different repetitions further strengthens both systems of centers as long as the repetition and alternation are natural and unifying, bringing the two systems of centers together into a larger and stronger system[15, pp. 165-172].

5. **Positive Space** - Positive space occurs when every center is whole and complete in itself and as part of the larger system of centers, with no leftover or meaningless space in the system. Every part contributes to the wholeness of the system and has a clear and definite place and responsibilities[15, pp. 173-178].

6. **Good Shape** - A good shape is a center which is made up of powerful, intense centers, which have good shape themselves. The elements required to make a good shape include a high degree of internal symmetries, bilateral symmetry, a well-marked or differentiated center, the creation of positive space around the center that makes it strongly distinct, and a sense of closure and completeness[15, pp. 179-185].

7. **Local Symmetries** - Local symmetries are those that are closely interlocked with individual centers or groups of centers. Local symmetries are not imposed by outside forces but must be developed through the natural development of strong centers. As one center is developed and intensified, there must be a another center local to it that serves to balance and compliment it[15, pp. 186-194].

8. **Deep Interlock & Ambiguity** - Living structures must be interlocked or connected with their surroundings, usually through a boundary structure that is an ambiguous zone belonging to both the center and its environment. This boundary unifies the center in its proper place within the larger system, allowing it to communicate and interact with the system[15, pp. 195-199].

9. **Contrast** - Contrast is difference and differentiation, without which a system has no structure. The strongest and most intense centers are made from true opposites, those that
essentially annihilate each other when they are superimposed or brought into the same space. Contrast produces living structure when it strengthens the unity and coherence of the system. When it does not unify, it destroys structure and life[15, pp. 200-204].

10. **Gradients** - Gradients follow as the natural response to slowly changing circumstances in space and time as a result of the adaptation that occurs[15, pp. 205-209].

11. **Roughness** - Roughness is the quality that results when the designer/builder is egoless, allowing the larger order of the thing to be relaxed and allows modifications to occur as a result of the local demands and constraints in different parts of the design[15, pp. 210-217].

12. **Echoes** - When echoes are present in a design, the smaller centers and elements that make up the larger centers are from the same family or type, containing deep internal similarities between them that bring unity and coherence to the overall design[15, pp. 218-221].

13. **The Void** - Living systems cannot be all detail and activity. There must be places of rest and stillness to balance the clutter and details. Larger empty spaces allow for large and slower-moving processes and activities to take place[15, pp. 222-225].

14. **Simplicity & Inner Calm** - Simplicity and inner calm occurs when everything that is unnecessary to the system has been cut out; all centers that are not actively supporting other centers are stripped out and removed, leaving only what is essential[15, pp. 226-229].

15. **Not-Separateness** - This is the quality of a system that is at one with its world and environment. It has become an essential part of that world, and the world is better for the system’s existence[15, pp. 230-235].

As noted above, these fifteen properties are deeply interrelated. For example, to understand alternating repetition, we must understand the things that are repeating. These things must themselves be strong centers, but how do we know that they are centers? They have a discernable good shape, and the things that are between those that are repeating must also have good shape and positive space. We also need contrast between the centers that repeat so that we can see that there are different things repeating. Figure 3.2 identifies the interactions and interdependencies between the fifteen properties.

Alexander’s fifteen properties describe geometrically-based relationships between elements (centers) of a complex system. These properties are also associated with dynamic structure-preserving transformations that inject and intensify these properties in a system under development. Thus, the properties are not just observable attributes of the end products of design, but are fundamental tools that can be used by the designer to mold and shape the unfolding system. As the fifteen Fundamental Properties define observable characteristics and attributes of a complex system, these fifteen Fundamental Transformations define the basic processes for differentiating and unfolding the design of complex systems. Alexander’s descriptions of these transformations are summarized below[16, pp. 77-79]:
THE INTERACTIONS OF THE FIFTEEN PROPERTIES

If property A depends on property B or we need property B for a complete understanding of property A, then we mark cell AB.

<table>
<thead>
<tr>
<th>Property A</th>
<th>Levels of Scale</th>
<th>Strong Centers</th>
<th>Alternating Repetition</th>
<th>Positive Space</th>
<th>Good Shape</th>
<th>Local Simmetries</th>
<th>Deep Inte洛克 &amp; Ambiguity</th>
<th>Contrast</th>
<th>Echos</th>
<th>The Void</th>
<th>Simplicity &amp; Inner Calm</th>
<th>Not Separateness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levels of Scale</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strong Centers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boundaries</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alternating Repetition</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positive Space</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good Shape</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local Simmetries</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Inte洛克 &amp; Ambiguity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contrast</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Echos</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Void</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simplicity &amp; Inner Calm</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not Separateness</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: Interactions of the Fifteen Properties[15, p.238]

1. **Levels of Scale** - Intermediate-sized centers are introduced to fill out the hierarchy of scales. There are two basic cases of application: 1) Some part of the system has been loosely defined is differentiated into smaller parts one level of scale smaller than the starting part; and 2) An existing large center is strengthened and made more coherent through the introduction of smaller parts (one level smaller) which act together with the existing large center in a stronger hierarchy.

2. **Strong Centers** - In a very fundamental sense, all of the transformations work towards this goal - strong centers. However, the primitive form of this transformation involves giving weight, definition, and/or distinction to a weak center that has emerged in the system.

3. **Boundaries** - The boundary transformation works on a center by defining and strengthening the interface between that center and its surroundings. If the boundary has not been established, it is defined; otherwise the boundary is strengthened and refined in order
to distinguish it from the center and its context and to make the boundary a clear demarkation between the enclosed center and the outside world.

4. **Alternating Repetition** - This transformation is a basic way of bringing structure, both static and dynamic, to a large, undifferentiated system. A repeating pattern is defined within the large center (which may be a boundary center) in conjunction with a second pattern of repeating centers that are interlocked and alternating with each other. In the strongest applications, the frequencies of the two repetitions will be different, producing (at least) a third “beat” repetition pattern at a harmonic frequency of the first two repetitions that strengthens the overall repetitions’ coherence and unity.

5. **Positive Space** - Positive space is created and strengthened within a system by creating new centers in the spaces between other centers. This has the effect of refining and strengthening the spaces between the existing centers and shaping those spaces in preparation for the development of other new centers that will further strengthen the system.

6. **Good Shape** - This transformation is applied to an existing system of centers, usually formed through the Alternating Repetition transformation. The weak centers that were formed through the Alternating Repetition transformation are made stronger through definition and refinement, filling the spaces within the repetitions with living structure.

7. **Local Symmetries** - This transformation strengthens a center or system of centers through the introduction of a symmetry local to the center or system. This symmetry may be introduced through mirroring the center across a naturally occurring axis or through the introduction of an axis of local symmetry. This transformation is often the first transformation applied to a newly differentiated center.

8. **Deep Interlock & Ambiguity** - This transformation is applied primarily in boundary zones where it is used to bind the distinctly opposing parts of the boundary into a tighter and more unified structure. This transformation unifies a system, and is rarely applied in the early stages of differentiation.

9. **Contrast** - The Contrast transformation is a refining and sharpening transformation used to increase the distinction between two types or kinds within a system. Opposing centers are refined at the same time in order to sharpen their polarity, allowing the two types to more effectively complement each other.

10. **Gradients** - This transformation is used to build transitions of size, character, etc. Centers are rearranged and/or introduced to create a systematically varying coherence between two or more larger centers within the system. The Contrast transformation has the ability to introduce order and coherence within very local as well as very large areas of a system.

11. **Roughness** - The roughness transformation uses intentionally irregular centers to create the most regular fit possible, from the large-scale view, for a given configuration.

12. **Echoes** - The Echoes transformation applies procedures, structures, and characters of certain centers within a system to others in the system. This helps to generate a “family resemblance” throughout a system that strengthens the unity and coherence of the system as a whole.
13. The Void - This transformation is the process of garbage removal during the design and life of a system. Areas which are relatively undifferentiated and are not necessary (in their current state) to the overall life of the system are cleaned out and surrounded by a well-defined boundary that is grounded in more differentiated structure.

14. Simplicity & Inner Calm - The Simplicity transformation is similar to the Void transformation in that it is a cleaning process. However, this transformation does not produce bounded zones of undifferentiated space. Instead, the Simplicity transformation removes unwanted or unnecessary centers, differences, and other sources of complexity throughout the system.

15. Not-Separateness - This transformation can be thought of as knitting a system into its environment. Modifications are made to centers within the system so that they gain subtle aspects of the context, and where possible, centers within the context are also modified to gain some of the substance of the new system.

As noted earlier in this section, a principle is something that is established as a standard or test used to measure, control, and guide practice[82]. The dual nature of Alexander’s properties and transformations suggests that, when taken together, they are fundamental principles for the design of complex systems. Viewed as properties, they are a value-based metric for evaluating the strength and weakness of a system’s design as well as providing constraints on changes to the system. As transformations, they serve as guides for adapting and evolving a system’s design in response to contextual changes and increased knowledge about the system and its environment. In order to make these principles more understandable to novice software designers, we have modified and compacted Alexander’s original descriptions into concise statements more directly applicable to software system design.

**Principle 1 Levels of Scale:** Complex systems have centers with a range of sizes that exist at a series of well-defined levels at definite intervals.

Levels of Scale is most often considered a visual phenomenon in a designed artifact: details in the artifact emerge or fade based on the viewer’s proximity to the artifact. When we look at a structure or system, the details we see and experience depend upon how close or far away we are from it. As we move closer, the larger structure extends out of our range of vision and is experienced less, while smaller details become visible and more defined. The opposite occurs as we move away from the structure: we no longer see the small details but the larger structure becomes more evident. Note the many levels of scale in the portico of the Rüstem Pasha Mosque in Figure 3.3. We can view the door and facade as a whole from a distance, and as we approach it more details emerge: the different patterned areas surrounding the door, the heavy jambs and ornate lintel, the panels and rails of the doors themselves. Move closer and you see more details in the decorative tiling and the intricate carving in the door panels. Each level represents a step
in an orderly progression of size and detail, and the elements at the same level share a common proportion and detail.

Similarly, at the lowest levels of a software program we have literal values, variables, operators, and programming language keywords. These are combined into expressions and statements. Groups of statements form blocks and functions, procedures, or subroutines that perform more specific and complex actions. We group these program fragments into larger constructs: modules and classes that encapsulate higher levels of complexity and functionality. Modules are combined to form programs and libraries of code, and programs are combined to form software systems that perform a variety of related tasks for the user.

This principle relates directly to the composition and decomposition of a system and to how efficiently “chunk” information. Software system designs are made up of components of many different logical sizes at various levels of abstraction and complexity. Salingaros showed that the base of the natural logarithm, the common scaling factor in naturally-occurring systems, is also an optimal factor for architectural forms[218]. Since we unconsciously perceive this scaling in the natural forms all around us, built forms that incorporate the same scaling are also perceived as natural rather than alien and out of place. From the perspective of software system design, differences in abstraction, complexity, or computational information content between any two elements in a system must be proportional to the distance between them using a consistent scaling factor. Natural levels of scale help make a system easier to understand, and as a result, easier to design and develop because the amount of information exposed at any level is related
to information at other levels by scales that are naturally and consistently recurring.

**Principle 2 STRONG CENTERS:** Important and/or critical elements orient and focus the understanding and application of a system, and these critical elements are refined and strengthened by the elements that surround and relate to them.

Architecturally, strong centers are the focal points of a building or region that initially draw our attention, and often our physical presence into themselves. Strong centers define what the purpose of a building is and how it should be used or occupied. In a house, public areas (e.g., living room, formal dining room) are strong centers that are highly visible. These rooms are situated so that they are easily accessible to guests. The accoutrements in these rooms should also be of a public nature. Less public rooms such as the kitchen or family room are not as visible to the guest. They are furnished and decorated according to the private comforts of the family as a whole, to form a strong center for the family to occupy together. The most private rooms in the house are fully separated from the public areas, and are strong centers for the individuals that occupy the rooms. These rooms welcome the individual warmly.

The towers of the temple at Angkor Wat in Cambodia, shown in Figure 3.4 illustrate the concept of strong centers. In this photo, you can see the bases of four other towers at the corners of the main temple building. Together, the towers focus your attention towards the middle tower, the largest and strongest center within the temple, which itself was a center within the encompassing city. This picture illustrates how centers are strengthened by the centers that surround them. The center tower is strengthened by the two towers next to it, the corner towers, the broad, paved approach to the building, and the arched entryways into the building. All of these elements draw you towards and into the heart of this important place.

Another interpretation of strong centers, within the domain of software design, is that of the key abstraction: the design decision encapsulated within a module, class, or other program structure. The need for this abstraction is created by the surrounding field of objects or modules within the system. The abstraction generates a set of smaller, dependent centers — the methods, data fields, etc. within the module or class that are necessary to implement the abstraction’s functionality within the system. These smaller centers are reflected outwards as well in the form of methods/functions in other classes or modules that use this strong center. The strength of centers within a software system is hierarchical, just as it is in physical entities.

Strong centers are the key abstractions, critical elements, and/or important decisions made at a particular level within a system’s design. Often, the most critical and important elements or aspects of a system are not fully understood (and may not even be identified) until late in the design process[87]. Alexander notes that strong centers evolve as the system as a whole evolves, and centers that were weak or hidden early in the development of the design are defined and strengthened by supporting elements[15]. Proponents of agile software development have
also noted this phenomenon, and recommend delaying critical decisions as long as possible to maximize the amount of information available to make such a decision[27]. Expert designers, based on domain knowledge, experience, and inference from known details, consistently insert metaphors or analogies into a design as placeholders denoting critical design elements that are not yet fully defined[24, 51]. Strong centers thus become the waypoints or anchors within the design of a system, enabling the designer to better navigate and understand the whole of the system.

**Principle 3 Boundaries:** *Individual components and design decisions should be distinguishable from others while also being unified within the system.*

Boundaries help one part of a system simultaneously isolate and reinforce other parts by creating defined areas of separation and connection. Correctly chosen, the smaller centers of the boundary region intensify and strengthen the larger centers they bound. An often overlooked but important aspect of boundaries is that they are recursive in nature. Because a boundary is itself a system of centers, each center within the boundary must have its own boundary made up of yet smaller centers, until fundamental entities in appropriate configurations are all that remain as centers. This recursive nature is evident in the intricate tile work shown in Figure 3.5. The boundaries are created from the solid blue tiles on either side of delicate, repetitive patterns, with each separate course itself a smaller-scale boundary. The strength of these boundaries is a result of the smaller centers they are made of as well as the proportions of the boundaries relative to the strong centers they enclose and the centers outside the boundaries.

Boundaries distinguish different design elements from each other while simultaneously providing the means for those different elements to connect and relate to each other. The use of
abstract interfaces in software design has a long history, and has been shown to be an effective tool for modularization and information hiding. Interfaces define the externally visible attributes of a software module, describing the functionality and information that the module provides while hiding the implementation details. Boundaries in software systems exist at many levels, from programming languages and their primitive constructs that separate the programmer from the hardware, to classes and modules that package function and information, function/method signatures, type systems, APIs, network protocols, and user interfaces. Boundaries delimit information chunks and provide a named abstraction or metaphor describing the attributes of the information they contain, allowing the designer to work with the abstraction rather than maintaining all of the details it embodies[90, 163].

**Principle 4 Alternating Repetition:** The dynamic behavior of a system is strengthened and stabilized by rhythmic, harmonious repetition among and between the parts and levels of the system.

All structures and systems exhibit some form of repetition, either in the structure or composition of the system or in the events that occur in and around it. Repetition occurs at all levels of scale: atoms, waves, music, buildings, and communities for example. However, simple repetition does not strengthen a system. Consider a loop structure within a program that does not provide a means to change the loop condition: once entered, the loop will execute infinitely,
preventing the rest of the program from executing. Similarly, if we construct an apparatus with a motor driven wheel that just spins without doing any productive work, we have repetition without purpose. The value of repetition within a system comes from the purpose and order it brings to the system. This order is strengthened when the repetition has a counterpoint or opposing repetition such that the two repetitions are synchronized in some way.

It is also critical that each of the repeated elements are not identical, but have modifications that are dependent upon the position and purpose of the individual element within the whole. The repeated elements are locally strong centers and have a well-defined purpose and meaning within the part of the system where they exist. In the fern leaf shown in Figure 3.6 we see alternating repetition at different levels of scale. The large leaf is divided into a series of leaflets, which are in turn, divided into smaller leaflets before being divided again into the smallest leaflets in the leaf. At each level the leaflets alternate on the main stem or vein of the leaflet, and this common structure is related at each level from top to bottom.

![Figure 3.6: Fern Leaf](image)

The individual parts of the system must interact in a consistent and regular manner if the system is to be stable and productive. A first design consideration from this perspective is that of identifying the equilibrium points - the situations where the system is in a steady-state oscillation with respect to the data and/or control flow across system boundaries, function calls and returns, user interactions, event handling, etc. If these nodes of interaction are not correctly identified, designing the timing of the system is difficult if not impossible.

Another example of alternating repetition in action is the interaction between software testing and the development of the software system, particularly when an agile process is used. Agile software development is generally test-driven, with unit, integration, and acceptance tests
commonly written before the actual implementation. Tests are run, code is written, tests are
run again, repeating this cycle until the system is complete.

Software systems are informationally dynamic: they take in information, transform it
computationally into new forms, and send the new forms out to users of other systems. Iteration
is one of the three fundamental types of program statements[31], but alternating repetition is
distinct from simple iteration. Rather than just repeatedly executing some part of a program
until some condition is satisfied, alternating repetition describes the rhythmic interaction between
two or more elements in a software system: the complimentary and cooperative interaction and
exchange of information. Scheduling algorithms, queueing systems, communication protocols,
and user interfaces are applications of this principle. Awareness of this principle helps expert
designers understand that transfers of information, execution control, and other dynamic aspects
of the software must be consistent and appropriate: requests must be properly fulfilled, input
and output must be balanced, and so on, if the system is to be dynamically stable.

**Principle 5** **Positive Space:** *Every element in a system must be well-defined, purposeful,
substantial, and a strong center at its own level of scale and place within the system.*

Alexander defines positive space simply: “...every part of space has positive shape as a center.
There are no amorphous meaningless leftovers” (emphasis added)[15, pp. 173-178]. In the visual
arts, positive space is the subject or foreground shape(s) or entities, and negative space refers
to the background or environment of the subject. Viewed in this light, Alexander’s definition
has profound implications: a living system is not only positive in and of itself, but it creates
and strengthens life in the context where it exists. In practical terms, this means that system
designs should only incorporate the elements necessary to accomplish the purpose(s) of the
system. Each of these elements, in turn, should contain only those parts essential to performing
the tasks of the element. Matisse’s cut-paper Blue Nude in Figure 3.7 illustrates this visually:
the pieces of paper (positive space) and the spaces between them (negative space) are intimately
interwoven yet distinct, injecting the image with life and strength.

Agile software development methodologies such as eXtreme Programming advocate deferring
work on potential future requirements and extra flexibility in favor of concentrating what is
needed at the present[27]. Change will occur over the life of a software system (including while it
is in development), and managing that change is critical. The Positive Space principle requires
that a system’s design and implementation are appropriate and complete for the requirements
at that point in time.

We can expand on this idea by noting that robust software systems are adaptable and
maintainable, and if a feature or requirement is discovered in the future, the system should be
able to accommodate the necessary changes with effort proportional to the size and/or importance
of the feature or requirement. It is rare that any such change is completely new. Rather, these
changes are usually related to existing parts of the system, and involve modifications to those parts. The positive space principle requires that, at a given point in the life of the system, all of the parts are well-formed and complete in themselves and according to their purpose within the system, and that there are no unused elements in the system. Poorly designed systems that have been “patched” together without any attention to this principle will be difficult to maintain because many parts of the system will not be in their “proper” places.

Principle 6 Good Shape: The strength of a particular element within a system depends upon its fitness to its purpose, its completeness, and that the element itself, its boundary, and the other system elements around it are all strong centers.

Like all of the fundamental principles found here, Good Shape is recursive in nature: a center or element with good shape is itself made up of elements that have good shape, continuing through levels of scale in the system until the simple, elementary parts of the system are reached. In a visual sense, these elementary entities are simple shapes such as squares, lines, arrowheads, triangles, spirals, stars, and S-curves. However, good shape does not emerge from random combinations of these shapes. Rather, good shape results from regular, well-ordered combinations of these elements to form more complex shapes, that are again combined to form even more complex forms.

Good shape results when each part is a fully developed center within the overall system of centers. Intense major centers are surrounded by intense minor centers that reflect the behavior
of the system. The Chladni figure shown in Figure 3.8 illustrates this. The intense center at the middle of the image is the post used to vibrate the plate, and the surrounding centers include the radial (curved) and diametric (straight) harmonic nodes and their intersections. The patterns produced depend upon the size, shape, and material of the plate, as well as the frequency and amplitude of the vibration. These modal patterns are critical in the design and construction of acoustic instruments such as guitars, violins, drums, and cymbals.

From the perspective of software systems, one example of these elementary forms is found in the structure of programming languages. Operators and expressions are combined to form statements, which are combined to form program segments such as functions and classes, depending upon the particular language. Libraries, and other compilations of programming resources provide a second tier of elementary forms: predefined data structures, operating system interfaces, user interface widgets, and communication protocols to name a few. While programming languages do have rules defining how these programmatic elements may be formally combined in the text of a program, there are often many different ways to write code to accomplish a given purpose. Sometimes tradeoffs are made between processing time and storage space; other times the readability of a piece of code is sacrificed for a simpler implementation.

In the realm of software system design, there are many places where this principle comes into play. Applying this principle at a higher level in the software development process, the development team itself should exhibit good shape if the project is to be a success. So how can we define good shape for a software development team? First, our “elementary entities” are the
people involved in the design and implementation of the system. Individually, each person must have the skills necessary to complete their responsibilities within the team. A programmer who does not know the language(s) used will negatively impact the effectiveness and productivity of the team. The team members must also have the ability and inclination to work together as a team. If the members of the development team do not have “good shape” the team will not meet this principle and will not be able to function efficiently as a team to meet the project goals.

Every part of the system is important: a chain is only as strong as its weakest link. Robust and sustainable software systems must be well designed and implemented from the most primitive elements through the large-scale system architecture. Good shapes in software are the basic design and implementation elements related to the problem/solution domain that are well-defined and understood, such as programming idioms, documented algorithms, domain-specific business rules, and design patterns. These basic elements are combined into more complex components that bring these elements into effective and efficient relationships, maintaining the clear definition of the smaller elements and fitting within the vision of the system as a whole. The structure and coherence of good shape enhances the understandability of larger components and allows the designer to use the same navigational tools to move through different parts of the system.

**Principle 7 Local Symmetries:** *Elements within a system should contain entities that are symmetrical and balanced within the larger element.*

Local symmetries are the essence of any dynamic system. Changes in the whole system are a cumulative result of the balancing of forces that occur at different places within the system. The existence of local symmetries provide the means to react to imbalances or stimuli and return the system to a new equilibrium zone. On the other hand, large-scale symmetry in complex systems appears forced and artificial, lifeless, and uninteresting. Because inputs to the system occur in specific local areas, the large-scale symmetry does not provide a means to resolve the resulting imbalance.

Taken as a whole, the Amsterdam canal houses in Figure 3.9 are not symmetrical. Individually, the houses are not perfectly symmetrical, either. However, each house contains many locally symmetrical elements, including window divisions, window arrangements (vertical and horizontal), dormers, and chimneys. These symmetries are repeated in similar but distinct ways across the group of houses, building and sustaining a feeling of symmetry and order in the group as a whole. Note that it is the local symmetries that make this contribution, not the existence of a large-scale symmetry.

Smaller, localized symmetries within a system arise from the development of related centers: as one center is strengthened, those centers that are adjacent or complementary to it are also strengthened in proportion to maintain the balance of that part of the system. These symmetrical
relations are not perfect - there are subtle variations imparted by the other centers within the system that are also related, but to different degrees and strengths. In software systems, these variations are commonly the result of changes to the structure or content of the information processed by the system.

The most important aspect of local symmetry is that it naturally arises when a center is strengthened. This is because centers do not exist by themselves - they are always made stronger within the context of other centers that must also be strengthened to maintain the balance of the system. Local symmetries are cumulative, giving the system balanced and harmonious structure at each level of scale.

There are many examples of local symmetry in software systems. The processing invoked by a function call should be proportionate to the data provided in the function’s arguments, and should be appropriate for the context in which the method is called. While it may seem that a program (or part of one) taking raw, unstructured data as input and producing refined, highly structured information might violate symmetry, but this is not the case. The raw data is only one input; the other input comes from the “knowledge” embodied in the algorithm and code used to structure and transform the data into more useful information. Data structures should reflect the data they hold and the manner in which it may be accessed. Specifications must reflect the requirements they satisfy, just as design elements must reflect the specifications and requirements they implement.
Pair programming is another example of local symmetry in a software development effort\cite{27, 252}. The driver and navigator roles are both responsible for designing and implementing code, but they are looking at the code from two different perspectives. The driver controls the keyboard, and is concerned with writing syntactically correct code that solves the problem at hand, while the navigator studies the code to ensure it fits within the goals and requirements of the system as a whole.

**Principle 8 Deep Interlock and Ambiguity:** *Strongly related components share well-defined boundaries that create coherence without generating unnecessary interdependence between the components.*

Deep interlock and ambiguity are characteristics of the boundary or interface between two related or collaborating system elements. The boundary unites the two distinct elements towards a common purpose while also maintaining the individuality of each element by becoming a zone that belongs to both elements and the context or environment in which those elements exist. The deep interlock results from a strong “contract” between the elements on either side and within the boundary. At the same time, the boundary also isolates the internal structures of one element from the other, hiding or obscuring the details behind the interface zone.

The dovetail joint shown in Figure 3.10 illustrates the Deep Interlock and Ambiguity principle in the interlock between the timbers of the two walls in this picture. The ambiguity results from the presence of the joint as an integral part of both walls at the same time. The alternating tapers cut on the “fingers” of the joint connect and secure the individual timbers into two distinct walls and the corner of the building. When this type of joint is used in fine cabinetry, the ends of the fingers are usually cut off and sanded flush with the opposing surface, leaving the ends of the fingers to appear to float in the panel. Because of the strength imparted by the tight interlock of the dovetail joint, it is commonly used in high-stress areas such as joining a drawer front to the sides and in making small boxes with relatively thin sides.

This principle addresses one of the most critical issues in software system development: the relationships between the programmatic parts of the system. On one hand, we want our programs to be efficient, which means designing and writing code that executes as fast as possible while using the least amount of system resources. Often, this requires that parts of a program use “intimate knowledge” of other parts of the program in order to achieve performance increases. On the other hand, we know that requirements, computer hardware, and other aspects of the system are going to change over time, and we need to create software that can easily be adapted to these changes. To accomplish this, we design and implement “modular” programs that separate the various subtasks into modules that hide the actual implementation behind a stable interface.
The Deep Interlock and Ambiguity principle captures and explains this dichotomy. Individual parts of a system must be just that - individual and distinct. As we see in the Positive Space principle, each element “must be well-defined, purposeful, substantial, and a strong center at its own level of scale and place within the system,” performing a unique and necessary function within the system. But they must also interact and function with other parts in the system if the system as a whole is to work. The system as a whole depends upon each part living up to its responsibilities.

**Principle 9 CONTRAST: Opposing forces define and strengthen system components and their boundaries.**

Light and dark, hot and cold, quiet and noisy; opposites help us comprehend and navigate the world around us. The world is full of opposites, and in very fundamental ways, it would not exist without them. The opposite electrical charges of electron and proton enable chemical reactions to occur and complex molecules to form. Elementary particles and their anti-particles enable nuclear reactions such as those that power the sun and stars. Night and day are basic divisions of time. Contrast often makes itself known out of necessity as a design develops, in the form of opposites that are functionally necessary.

Contrast helps to define and strengthen boundaries within a system by clearly demarcating differences between related elements. By making an element’s boundary more distinct, the
element itself becomes stronger and more clearly defined in relation to the context or environment where it exists and functions. Individual attributes and behaviors are easier to identify, and the internal structure becomes more robust.

The stone garden shown in Figure 3.11 also illustrates the balance brought by contrast. The large rocks jutting out of the rippled sand become strong centers, foci for rings of ripples, and dividing the field into smaller and more interesting pieces. These pieces are part of the garden as a whole, but bring balance to the whole as individual entities within the garden that compete for our attention.

![Figure 3.11: Stone garden of Komyozenji Temple in Dazaifu](image)

Contrast is generated and strengthened both from within an element and on its interface with its surroundings. Internal contrast is increased when we focus on defining and strengthening the internal structure and behavior of the element. External contrast is increased by defining and strengthening the interface or boundary that separates the internal structure from its context. It is important to note that these two operations are closely related. As the boundary is defined more specifically, the internal structure and behavior must be modified to accommodate these changes. Similarly, when the internal structure is refined, it may be necessary to modify finer details of how the element interacts with its surrounding elements in the system.

Input–output, producer–consumer, and transmitter–receiver are examples of opposites common in software systems, and without these opposites, it is difficult, if not impossible, to determine if the system is doing anything useful. Often requirements define contrasting needs
or goals for a system: space utilization vs. processing time, for example. Different users or stakeholders in a system may also have different expectations and requirements for a particular system of unit of functionality. Expert designers look for contrast and difference to identify critical decision points (strong centers) within a system and strive to achieve balance between the competing perspectives.

**Principle 10 Gradients:** Change is best managed through gradual transition and refinement.

Change is a necessary part of all complex, dynamic systems, and how change is managed will often determine the success or failure of the system. When change is allowed to occur suddenly, without warning or anticipation, the results can be disastrous as the system or the user absorbs the full shock of the change at once. This shock can destabilize the system to the point of failure. Alternatively, when change is anticipated and spread out over time and/or space, it is much easier to manage and understand, and much less likely to destabilize the system.

Gradients also result from natural growth processes. The nautilus shell Figure 3.12) provides several nested gradients that result from the growth of the mollusk that makes its home in the shell. The outer shell is a gradually expanding spiral filled with interior walls that are themselves expanding radius curves. The proportions of a nautilus shell as it expands approximate a Fibonacci sequence defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$, a recurrence found in many different settings in nature.

![Figure 3.12: Nautilus shell cut in half](image1)

68
Gradients are an important part of designed entities for both aesthetic and structural reasons. As transitional devices, they allow a designer to create directed transitions from one part of an entity to another. The gradient leads the mind through a controlled and gradual transition from one important part of the system to another. Without a gradient, the break is abrupt, requiring the mind to quickly reorient itself to the new configuration. Structurally, gradients allow the designer to concentrate and diffuse the information content, organization, and physical strength within a system. This allows the system to be more resilient and able to handle even sudden changes with less chance of instability.

Normal computation in a software system proceeds smoothly along predictable paths defined by the implementation. Data and control flow through a system should occur smoothly to enhance the readability and maintainability of the system. A user’s interactions with the system should also proceed in a logical and orderly fashion in order to support the understanding and productive use of the software. Abrupt changes introduce discontinuity and often become bottlenecks that hinder the performance of the system as a whole. Expert designers seek to maintain smooth and understandable transitions by developing an understanding of the problem domain and of the processes the software is intended to model.

**Principle 11 Roughness:** Robust and reliable systems must expect and accommodate change and variation.

Roughness is the product of an adaptable design and design process that allow the larger order of the system to emerge and become whole based on the requirements, constraints, and other demands that are local to different parts of the system. It does not mean that a system is designed to be unpredictable or constructed in a sloppy manner. On the contrary, this principle requires that the system be precisely what is needed, nothing more and nothing less, even though the details of those needs may not be fully known or understood when the system is constructed.

Roughness is not the result of errors, inferiority, or inaccuracy. Rather, it is the subtle and precise adaptation of a common shape or configuration to meet the needs of a particular local situation. Figure 3.13 shows the intricately fitted stonework of an Inca wall, where each stone has been cut and shaped to fit a particular spot in the wall. The different sizes and shapes of the stones contribute to the strength of the wall even though it was constructed without any kind of mortar or cement: the varied shapes are integrated into a solid and durable structure.

Large software systems cannot be robust and reliable unless they incorporate some roughness that allows the systems to be able to adapt and relate to current conditions rather than being rigidly and inflexibly built for some “ideal” environment. As complex as software systems are, the environments in which they operate are usually far more difficult to quantify or predict. Differences in computer hardware, communication delays, equipment failures, and resources shared with other programs and systems are just a few examples of the environmental complexity.
where software systems execute. Roughness and variability provide the “slack” that allows a system to bend without breaking, absorbing the shock of unpredicted events without failure.

Roughness is also essential in the process of developing software. Regardless of the model employed, the development process is a learning and exploring journey. Attempting to rigidly define the process that will be used to create a software system ahead of time is an invitation to failure. The development process must be open to modification and refinement as individuals come and go, requirements and technology change, and the software is actually implemented and tested. Agile development methods such as eXtreme Programming incorporate short design and coding iterations, constant communication with the customer/users, and continual planning and replanning as core values[27]. Decisions that may have significant consequences for the system should be deferred as late as possible to enable the discovery of fine details that influence such decisions. The Spiral Model also incorporates continual discovery, reevaluation, and risk assessment, again to provide designers and developers with the maximal opportunity to understand the system before making commitments that will be difficult to change[30].

**Principle 12 ECHOES: A complex system is strengthened and unified by common themes that echo throughout its depth and breadth and by echoing the world in which the system exists.**

Alexander speaks of *Echos* as family resemblances, the underlying and pervasive similarities that run through a system. This common thread or theme helps to unify and strengthen the system. When a system echoes its surroundings it becomes more unified with that environment. The system does not just exist in that environment, but becomes an integral part of it as a larger system emerges that incorporates both the new system and the context where it exists. Robust and beautiful systems echo the foundational structures that support the system, as well as the processes and forces that create and sustain the system.

Echoes are the common threads and themes that run through complex systems. We find echoes in great music: chord progressions, rhythms, and melodic snippets that trace their way
through each movement of a symphony. We find echoes in great architecture: shapes, angles, and configurations that repeat and recompose themselves in smaller and larger instances. The basket shown in Figure 3.14 contains echoes of the culture and geography of the Squamish people. Living along the southwestern coast of British Columbia, the Squamish were bounded by water and mountains. Echoing through the basket, the rippling cording reflects the rolling waters of Howe Sound and the rivers and streams running through their native land. The strong, tightly-woven bands recall the granite mountains and massifs that permeate this area of Canada.

If a system does not have effective internal and external echoes, it is likely that important functional requirements have been overlooked or misinterpreted. Internal echoes in a software system help guide users through the tasks the system is made to accomplish, and missing internal echoes are a sign of missing tasks, cues, or navigational elements that support the user’s productivity. External echoes unify a software system with the execution environment with system features and attributes that echo and reinforce the system’s relationship with that environment. If these echoes are absent, it is likely that relationships between the system and its environment have been broken.

At their core, software systems echo the real world. Intelligent systems model human
perception, learning and decision-making. Game makers strive for environments with more realistic visual textures, animation, and sound. Personal finance software allows users to write checks on the screen and make entries into a check register. By using echoes and metaphors, learning curves are minimized because the systems rely on users’ familiarity with real world entities and processes. As more and more people have become accustomed to using computers, common and familiar actions echo across software applications and platforms. One of the most common tasks for a computer user is saving a file. The **CONTROL-S** key combination has become an almost ubiquitous shortcut for this operation. We are caught off guard when we encounter a program that does not use this combination for that purpose (often losing a file as a result).

**Principle 13 The Void:** *Empty or still space is essential for growth and new development within a complex system.*

Systems without empty, unstructured space become cluttered and confused. To the observer, *The Void* adds a place of rest, a nest for growth, and a boundary to connect new development with the existing system. Systems must have a space of tranquility within their busy and changing environment to be an anchor and allow the system’s energy to coalesce and focus. One key attribute of a void within a system is its boundaries with the surrounding elements. These boundaries define the empty or still place, separating it from the active parts of the system and providing the means for using the region when needed.

Saint Peter’s Square in The Vatican, shown in Figure 3.15 is a void brimming with potential energy and waiting to be filled. This space was designed and constructed in the 17th century to allow as many people as possible (at least 250,000) to see the Pope and hear his blessing. This space is intended to invoke feelings of awe, strength, and permanence through its size and the massive colonnades surrounding the ellipse and the entrance to Saint Peter’s Basilica.

The presence of void space within a system is often not an explicitly stated functional requirement, although the space is absolutely necessary for the long-term health and sustainability of the system. The space within a barn is a product of the building’s dimensions, roof type, and other structural features, factors that are all described in the plans for the building. The void space appears as a result of fulfilling the stated requirements.

Software systems operate within the constraints dictated by hardware and operating systems. One of these restrictions is often the amount of physical and/or virtual memory available to the application. Software systems must make efficient use of the resources allocated to them. As the system accepts input for processing, memory within the system’s allocated space is filled with data. As the need for that data is exhausted, memory becomes available for reuse, provided the system was designed to efficiently reclaim and reuse it.
Another example of the Void within a software system is the user interface. Business systems ordinarily require user data input of some kind: there must be empty spaces to enter that data. Systems that allow creative endeavors (e.g., word processor, image, sound, and video editing systems, etc.) must give the user a canvas to work on. As the user is given more freedom to be creative, the canvas must have less predefined structure.

**Principle 14  Simplicity and Inner Calm:** *Always do the simplest thing that could possibly work, removing the extraneous and retaining only the essential.*

Simplicity and Inner Calm represents a commitment to keeping a system as simple as possible, retaining only those elements that are necessary to achieve the goals and requirements of the system. This principle is not only about keeping a system simple. Simplicity does not imply a lack of complexity; it only requires that everything in the system is essential to the purpose of the system. It is also important that each part of the system serves a definite and known purpose, actively supporting other elements and ultimately the user’s or customer’s needs for the system. Inner calm is the result of all of the parts of the system working together in smooth harmony to meet these goals.

We can see this kind of simplicity and inner calm in the built-in furniture shown in Figure 3.16. There is nothing extraneous here - the furniture is even built into the wall so that it does not protrude into the calm of the room. Every bit of space is used so that nothing is wasted, and the hardware and ornamentation is just enough to fulfill the functional needs while clearly indicating the interior space available and how the space is accessed.
The most appropriate application of this principle is to include in a system only those elements that are essential to the system at that time. However, over the development and lifetime of any complex system, there are parts that become obsolete or are replaced by newer elements, and these unused parts must be removed to maintain the life of the system. This principle helps to identify unused or duplicated elements, obsolete attributes or behaviors, conflicts, and other things in the system that add complexity without producing any value from the system.

This process can be one of the most difficult tasks in design because software developers have the habit of trying to plan for future enhancements or functionality that have not yet been requested. Often this is done under the guise of making a system robust or redundant, but added complexity that does not accomplish anything actually weakens the system since it is one more part of the system that must be maintained and checked when modifications are made. Frequently these “future-looking” elements are ignored as development and maintenance continue, rearing their heads only when some modification exposes some aspect of the added elements in a negative way.

Novice software developers often make design problems more difficult than they actually are as a result of inexperience and a lack of domain knowledge[9]. Much of the design and implementation of software systems is related to the competition with other software for
scarce system resources such as processor time, communications bandwidth, and storage space. Extraneous and unnecessary functionality can wastefully consume these resources, and should be avoided in robust systems. Furthermore, design elements should be as simple as possible to meet the given requirements. Adding unnecessary complexity can make a system harder to understand, which translates to greater maintenance and training costs. Expert designers seek to maintain simplicity and consistency throughout the system.

**Principle 15 Not-Separateness:** A sustainable system is one with its environment, not separate from it.

In many respects, this principle is the essence of the concept of centers and the systems they form. A “center” in complete isolation cannot serve any purpose or meet any need because it is not connected to anything else. Not-separateness means that a system is deeply connected to the world where it exists, that the system and its environment are one with each other. This principle primarily applies to boundaries since these are the systems of centers that connect the inside to the outside. The feature that exemplifies this principle is the lack of abruptness in how the system fits into its context: it does not stand out, it fits in like it belongs there, because it does.

Consider the stone path shown in Figure 3.17. The stones seem to emerge from the ground just enough so that you know this is a path, a reliable and safe way to cross the moor. It looks natural, as if it has always been there and always will be. The path is a strong center, its strength coming from its boundaries, the good shape and positive space of the stones, and the roughness of their fit with respect to each other and to the land. The boundary between the path and the land it rests on is rough and seemingly random, shaped by the stones themselves as they appear to grow out of the earth. The bare earth merges with the vegetation of the moor through gradients of growth and connection. Without the moor, however, this path has no purpose and no place to exist, and in this place it gains most of its strength from its connectedness with the moor.

In the design and implementation of a software system, each part of the system should be essential to the overall purpose and requirements and fit in its proper place within the system. Everything works together efficiently and effectively to accomplish the system’s goals. If we shift our focus to view the system under development as a single center within the larger system of centers that is its environment, it must exist cooperatively with the other software applications running on the same platforms[108]. From this perspective, all of the principles that apply within our system now apply to our system’s relationships with those with which it shares a common environment.

In the larger human context, a software system is often one of many resources at a user’s disposal, and one of many entities competing for the user’s attention. Designers must be aware of
the impact their systems have on their users, and should design systems that integrate smoothly and naturally with the human environment in which they are used\[41\].

Although it is listed last, the Not-Separateness principle is perhaps the overarching goal of all of the principles. A system that exhibits this principle is internally complete, with every part being necessary and sufficient. It is also complete within the external environment or context. Each part of the system is distinct, unique, and purposeful, but is also supportive of the local wholeness and the overall unity and functionality of the entire system. The system is an integral part of its environment as well and this larger system is completed by the thing we have designed and built.

### 3.3.2 Patterns of Connections and Meanings

Design patterns and pattern-oriented instruction have also been shown to help students learn and use expert design skills such as analogizing, problem decomposition, and solution construction\[185, 186\]. For the purposes of this research, we adopt Alexander’s definition of a pattern:

*Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must create it. It is both a process and a thing; both a description of a thing which is alive, and a description of the process which will generate that thing.\[17\]*
Patterns are distinct from plans. A plan describes a method for doing something, and following the plan will always result in the same thing being done, within the scope of the plan. A pattern is less and more: less in the sense that it is not deterministic as a plan is; and more because a pattern describes a set of possible resolutions to a problem that are dependent upon the particular context of that problem. The pattern foreshadows the product: it is the rule for making the thing, but in many ways it is also the thing itself. A pattern is more than just the rule, it encompasses the literature that describes the rule and helps it unfold and be used. Alexander explains this relationship:

*These patterns in our minds are, more or less, mental images of the patterns in the world: they are abstract representations of the very morphological rules which define the patterns in the world.*

*However, in one respect they are very different. The patterns in the world merely exist. But the same patterns in our minds are dynamic. They have force. They are generative. They tell us what to do; they tell us how we shall, or may, generate them; and they tell us too, that under certain circumstances, we must create them.*

*Each pattern is a rule which describes what you have to do to generate the entity which it defines.*

In order to help students understand and use the fifteen principles described above, we have written these principles as patterns. Since a principle is a rule that captures the intrinsic nature of a thing, and the principles we have defined also capture the strategies of expert designers, it is a straightforward transformation from each principle to a corresponding pattern. Based on Alexander’s definition of a pattern and common forms and content of software patterns, we developed a pattern structure for transforming our design principles into patterns. The elements within this Design Principle Pattern structure are shown in Table 3.5.

We use this pattern structure as a template for presenting the fifteen design principles in the \( P^3F \). Use of this provides the novice designers with supplemental contextual and relational information that helps explain how, when, where, and why each principle is applicable. In addition to providing a basis for design decision-making, these principle patterns also provide an abstract structure for the developing system to help facilitate navigation through the design space through the use of the pattern structure.

### 3.3.3 A Process for Decision-making and Movement

The third part of the \( P^3F \) is a structured but general design decision-making process. Design in general, and software system design specifically, requires practitioners to apply critical thinking and reasoning abilities, fundamental skills college graduates are expected to have developed.
Table 3.5: Design Principle Pattern elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principle Name:</td>
<td>The name of the design principle and a one-sentence statement that expresses it. The name is a metaphor representing the principle’s structure and generative quality.</td>
</tr>
<tr>
<td>Intent:</td>
<td>The intent of the principle identifies why the principle is important.</td>
</tr>
<tr>
<td>Forces:</td>
<td>A design principle exerts forces on an emerging design. Also incorporates the reasons for using the principle and its benefits and advantages.</td>
</tr>
<tr>
<td>Counterforces:</td>
<td>The environmental/contextual pressures and constraints that work against this principle. Design decisions represent a balance between the external forces and the counterforces.</td>
</tr>
<tr>
<td>Metaphors:</td>
<td>Alternative names that capture key aspects or perspectives of this principle.</td>
</tr>
<tr>
<td>Related Principles:</td>
<td>A design principle does not exist in isolation. A design decision based on one or more design principles results in a new situation where additional decisions must be made, based on other design principles.</td>
</tr>
<tr>
<td>Discussion:</td>
<td>A longer explanation of the principle that includes additional details about forces, counterforces, and intent as well as rationales for when and where the principle should be applied.</td>
</tr>
<tr>
<td>Examples:</td>
<td>Familiar or common instances of this principle to build association and understanding.</td>
</tr>
</tbody>
</table>

A fundamental connection exists between design thinking and critical thinking, a connection we used to adapt Alexander’s *Fundamental Differentiating Process*[16] into a tool help novice software designers make effective design decisions. Before introducing this tool, we discuss this important connection between design and critical thinking.

### 3.3.3.1 Design and Critical Thinking

In software engineering terms, software design is the group of activities that transform requirements and specifications for a software system into artifacts that communicate the structure and behavior of the system to the developers and programmers that actually implement the system. Design is “complete” when a complete model of the system (or some part of it) is ready to be implemented[204], but what this means in practice is dependent upon the development methodology used on a particular project.

Kazmierczak shifts the emphasis away from the objects and artifacts of the design process and onto the meaning that is created and communicated by those artifacts[150]. Rust compliments this definition of design by noting that the artifacts of design must embody the tacit knowledge of the designer in a manner that makes that knowledge accessible to the receiver[217]. Jones
states that the process of designing is “the designer’s way of discovering what he knows, and what he does not know, about this new thing he has promised to invent, and to integrate it into the world as it is.”[144] Knowledge acquisition, sharing, and integration are critical activities in the design of large software systems[250].

However, the process of designing is not only logical and rational, it is also creative and intuitive[78, 144, 221]. It is here that the connection with critical thinking begins to emerge. Quellmalz identified six key strategies of critical thinking[205]:

- Identifying and defining issues and problems.
- Determining the kind of information that is relevant to solving the problem or resolving the issue.
- Gathering, judging, and connecting the information.
- Generating hypotheses, constructing arguments, making inferences.
- Testing hypotheses, making counter-arguments.
- Evaluating the results, possibly revisiting earlier stages of the process.

These strategies are employed in a dynamic and metacognitive process that includes planning, monitoring, reviewing, and revising. Similar problem-solving strategies have been identified by Ackoff[3], Johnson-Laird and Wason[143], Polya[202], and Simon[230], among others. Intuition and insight are required to identify the actual problem that needs attention and what information is relevant to that task. Arranging and rearranging the relevant information in order to generate alternative solutions to the problem often demands creativity.

Design problems are usually “ill-structured”[228] and even “wicked”[58, 209] because they rarely have precisely prescribed goals or objectives, and the means of solving the (apparent) problem is not obvious[213]. Compound this with constantly changing technology and user requirements, and it is easy to understand how software system design becomes a very difficult problem. Software designers must continually refine and reformulate the problem(s) at hand, searching out new knowledge that may be applicable to the situation at hand, proposing alternative solutions based on the new and existing knowledge and experience, and learning from the evaluation of these proposed solutions [159]. The ability to objectively look at a problem from many different perspectives and to formulate and evaluate potential solutions that integrate those different perspectives is a necessary quality of the best software designers[189].

There are many similar but different ways of describing the process a designer follows when working on a design problem, all dependent upon the particular point of view of the researcher, the level of detail used, and the purpose of the process model. Takeda, et. al. have analyzed the reasoning that occurs in the course of a general design cycle for a software system[240]. Vaishnavi and Kuechler extended this analysis to explain and interpret the knowledge generated.
in a design effort and apply the cycle specifically to information systems design science research leading to the general design cycle framework illustrated in Figure 3.18[244].

![Figure 3.18: Reasoning in the Design Cycle][244]

In this model of design, abduction is a kind of theory-building or inference to the best explanation. Abduction is a creative form of inference that goes from data describing something to a hypothesis that best explains the data[146]. Abduction has been described as modus ponens turned backward, inferring the cause of something, generation of explanations for what we see around us, and inference to the best explanation[54]. Examples of abductive processes include medical diagnosis, story understanding, vision, and creative design are all abductive processes. Deduction, on the other hand, allows deriving \( b \) from \( a \) only where \( b \) is a formal consequence of \( a \). In other words, if \( a \) implies \( b \) and \( a \) is true, then \( b \) must also be true. Abductive reasoning processes are needed when design situations are poorly defined or understood and the design space must be moved or extended. The new configurations that result from creative design processes cannot be deduced from what is known[35, 70].

Tesar suggests that designers work from a particular and predominant frame of mind on a particular problem or part of a problem, illustrated in Figure 3.19. This mental perspective influences how the designer approaches and solves the problem at hand[241].

Software design differs from other design fields in its relationship with the physical world. Most designers create artifacts that will have some type of physical form such as a building,
automobile, or assembly line. The result of designing a software system is a complex set of instructions specifying sequences of operations executed by some computational device to achieve some (usually non-computational) goal. Despite the difference in the medium of design, effective software designers apply much the same thought processes as their counterparts in other design fields[73]. Like other design disciplines, they must also work in a complex environment that extends far beyond the individual, the development team, and the project at hand. Social, organizational, economic, and political factors all influence the designer and the design process as illustrated in Figure 3.20. Understanding the complete design problem, efficiently managing that understanding, and making reliable decisions about the design problem and process for solving it are all critical aspects of the expert designer’s cognitive toolkit.

Rather than rigidly applying a single methodology or approach, the expert designer remains flexible and adaptive, opportunistically using the information at hand to refine the solution and development path[120]. Intermediate forms and hypotheses represent the designer’s knowledge at a particular point in time and also expose new aspects of the problem and solution[163]. Inflexible design methodologies can severely constrain the designer’s ability to adapt or change their approach when faced with new information and understanding.
### 3.3.3.2 Design and Living Processes

Alexander defines living processes in the world of building as “any adaptive process which generates living structure, step by step, through structure preserving transformations.”[16, p. 204] This process is not one of trial and error but the emergence of a coherent whole that already existed but was latent until it was unfolded through the structure preserving transformations. Alexander also lists the following ten characteristics of living systems, features that must be present in the design, development, and application processes of living systems[16, p. 225].

1. A living process is a step-by-step adaptive process, which goes forward in small increments, with opportunity for feedback and correction at every increment.

2. It is always the whole which governs, in a living process. Even when only latent, whatever greater whole is latent is always the main focus of attention and the driving force which controls the shaping of the parts.

3. The entire living process - from beginning to end - will be governed and guided and moved forward by the formation of living centers in such as way that the centers help each other.

4. The steps of a living process always take place in a certain vitally important sequence, and the coherence of the results will be dependent to a large extent on the accuracy of this sequence which controls unfolding.

---

**Figure 3.20:** External factors affecting the software design process[39]
5. Parts which are created during the process of differentiation must become locally unique; otherwise the process is not a living process. This means that all repetition is based on the uniqueness of the locally shaped parts, each adapted, by the process, to its situation within the whole.

6. The formation of centers (along with the sequence of their unfolding) is guided by generic patterns which play the role of genes.

7. Every living process is, throughout its length and breadth, congruent with feeling and governed by feeling.

8. In the case of buildings, the formation of the structure is guided geometrically by the emergence of an aperiodic grid which brings coherent geometric order to built form.

9. The entire living process is oriented by a form language that provides concrete methods of implementing aperiodic structure through simple combinatory rules.

10. The entire living process is oriented by the simplicity transformation, and is pruned, steadily, so that it moves towards formation of a beautiful simplicity.

Of course, one problem that exists is how to define what a “good” and “living” software system is, what it looks like in code, what it looks like while executing, and how it appears to users. We can expand this further by asking who or what are the users of a software system? Are users restricted to the human beings that directly interact with the system? This is the usual view, but in today’s distributed and service-oriented software architectures, that distinction is becoming blurred. Often, the “user” of one system is some part of another software system that is making use of the functionality of the first system. Think about the major elements of a retail or e-commerce system: you have the human-user interaction system, often driven by a web server. There is a separate database or data warehouse system that is responsible for maintaining all of the necessary information; and there is the business logic part of the system that translates between what the user wants to do and how the data to accomplish those needs is stored and organized. The average user does not interact directly with the database - that is mediated by the user interface and logic systems - so those systems are actually the “users” of the database.

### 3.3.3.3 Alexander’s Fundamental Differentiating Process

There are also many “rules of thumb” that are taught as methods to design and implement better software systems. Types and class names, for example, should be nouns or noun phrases that identify a general category of things or objects. Method and function names should be verb phrases that describe the functionality that piece of code provides. Variable names should be semantically consistent with their context and data type. Table names in database
systems should reflect the association that ties together the table’s data. Code should be consistently indented to make it more readable, and should be well commented as an aid for future modification. Business rules and constraints must govern the external functionality of a system and may also constrain internal implementations to some degree (e.g., accounting procedures and tax laws). While these rules of thumb have significant value in their specific areas, they are not serviceable as design processes. For that, we must look to something more basic.

Alexander’s *Fundamental Differentiating Process* is one such process[16]. Differentiation is the creation, strengthening, or refinement of a center or element within a system with the goal of moving towards a more complex, developed, and coherent structure. Differentiation is achieved by applying the transformational aspect of one or more of the fundamental design principles to the system as a whole. In the early stages of design, the differentiation and refinement will be devoted primarily to developing knowledge about the desired system, including the desired outcomes of the implementation, the operational environment, and the human and organizational context of the system. As the process continues, more and more of the differentiation effort will be directed towards the system itself, although the external factors and influences must always remain part of the process. We begin by showing the process in its original form:

1. At any given moment in a process, we have a certain partially evolved state of a structure. This state is described by wholeness: the system of centers, and their relative nesting and degrees of life.
2. We pay attention as profoundly as possible to this wholeness - its global, large-scale order, both actual and latent.
3. We try to identify the sense in which this structure is weakest as a whole, weakest in its coherence as a whole, most deeply lacking in feeling.
4. We look for the latent centers in the whole. These are not those centers which are robust and exist strongly already; rather, they are centers which are dimly present in a weak form, but which seem to us to contribute to or cause the current absence of life in the whole.
5. We then choose one of these latent centers to work on. It may be a large center, or middle-sized, or small.
6. We use one or more of the fifteen structure-preserving transformations, singly or in combination, to differentiate and strengthen the structure in its wholeness.
7. As a result of the differentiation which occurs, new centers are born. The extent of the fifteen properties which accompany creation of new centers will also take place.
8. In particular we shall have increased the strength of certain larger centers; we shall also have increased the strength of parallel centers; and we shall also have increased the strength
of smaller centers. As a whole, the structure will now, as a result of this differentiation, be stronger and have more coherence and definition as a living structure.

9. We test to make sure that this is actually so, and that the presumed increase of life has actually taken place.

10. We also test that what we have done is the simplest differentiation possible, to accomplish this goal in respect of the center that is under development.

11. When complete, we go back to the beginning of the cycle, and apply the same process again.

There are three general ways that the differentiating process may be applied, related to the recursive nature of the fundamental principles themselves and to the iterative structure of the process. The simplest application is a single iteration of the process. A single pass means that the system is at its optimal and “final” state when the process is completed and that no additional changes are necessary. A second way the process may be applied would be in a strictly linear fashion, focusing on a single center in the system until that element is fully developed. Finally, the process may be applied in a nested hierarchy of steps, with each step being a separate instance of the process. This allows opportunistic movement and development throughout the system while maintaining navigational anchors through the hierarchy of steps.

3.3.3.4 Unfolding Emergent Design

While the Differentiating Process describes the active first-person process of designing and creating living structure, the Unfolding Process is the phenomenon we observe as the differentiating process is carried out in a particular context or environment. Another way of thinking about the relationship between differentiating and unfolding is that unfolding the design is what we are doing, in context, and differentiating is how we are doing the unfolding, e.g., making decisions about which part of the system to work on and which transformations to apply at each step in the process. As the fifteen structure-preserving transformations are applied in context, the design and structure of the new system unfolds or emerges and becomes more and more finely defined.

In order for the differentiation to unfold living structure, we must have the proper mindset, which includes an understanding of how we will approach the process, what factors will guide our decision-making, and how we view the developing system as well as the existing context or environment where that system will “live.” Alexander describes four conditions that must be present for unfolding (and the differentiation that causes the unfolding) to occur[16]:

1. *Step-by-Step Adaptation:* The process, whether large or small, must be step-by-step, and gradual. Each part of the environment, at every stage of its planning, conception,
and construction, must evolve, be developed step-by-step. The form must be created step-by-step, each step being an adaptation in which things get fitted more and more closely to a harmonious whole.

2. **Feedback:** To guide the adaptation, at each step in the process there must be a continuous and relatively immediate feedback about whether what has been done is a living structure in sufficient degree. In human society this requires as a minimum a common shared understanding of “life”. The process is then capable of adapting to this feedback, instantaneously, so that what has life can be kept and what does not have life will be rejected — with agreement — all while the process is going on.

3. **Unpredictability:** To make the adaptation successful, the process must be relaxed about the unpredictable character of where it goes. Unfolding cannot occur except in a framework which allows the whole to go where it must go. The dire modern passion for planning and advance control must be replaced by an attitude which recognizes that openness to the future, and lack of predictability, is a condition for success. It must be alright for the thing to become whatever it becomes, under the influence of adaptation and feedback, even though one does not know, in detail, what that thing is going to be.

4. **Awareness of the Whole:** Fourth, and this is the most difficult for us, there must be an ever-present awareness of the whole, throughout the process. For the adaptation to allow wholes to unfold successfully, the unfolding must take place within a framework of true awareness of the whole.

The sequence of steps in unfolding a design is critical to the success of the overall design. Consider a simple, two-step process of laying out a house and garden on a lot. If we place the house first, the garden must be “squeezed in” to the left over space, even though the garden is a vital component in the living structure of the house/garden system. Yet this is the common-sense approach frequently taken. However, if we locate the garden first, in a place where it will receive the proper sunlight and serve as a quiet and peaceful place for the residents of the house, then place the house so that it supports and compliments the garden, the result is a stronger whole with more living structure.

We can find a similar situation in software system development. A customer wants to develop a distributed, customer-centric system for order placement and tracking. The most common solution is to use the LAMP (Linux, Apache, MySQL, PHP) architecture. However, this presents problems if the customer is already committed to another operating system, database system, or inventory management system. Making the decision without considering the implications is like placing a garden in the corner of a lot adjacent to a busy, noisy intersection because that is the only space left after locating the house on the lot.
3.3.3.5 The $P^3F$ Differentiating Process

As a part of the $P^3F$, the Fundamental Differentiating Process provides a template for identifying and iterating over generative design decision sequences that allow a system’s structure, behavior, and interaction with its environment to unfold as a coherent whole that smoothly integrates with its environment. With each new iteration, we refresh the view of the system as a whole, incorporating the changes made in the previous iteration only if they strengthen the system as a whole. We also evaluate the current state of the system using the design principles as guide to identify the strengths and weaknesses that indicate the part of the system that need the most immediate attention.

In order to relate the process more closely with software system development, we restate it in more familiar terms:

1. At any give time in the design process, we have a certain partially evolved state of a software system. The state is described by the system’s wholeness: the system of centers, their relationships and nesting, and their fitness for their places in the overall system. The system is characterized in terms of the problem and solution domains: what solution must be in place to resolve the problem the system will address.

2. We pay attention as much as possible to this wholeness: the global, large-scale order of the system. This global order has visible components that are already under intentional development as well as latent components that are not yet visible, but are necessary to achieve the system’s goals.

3. We try to identify the sense in which the system is weakest as a whole, in particular looking at the coherence and completeness of the system as a whole with respect to the goals and requirements viewed through the lens of the fundamental design principles.

4. We look for the latent centers in the whole. These are not those centers that are robust and exist strongly already; rather they are latent centers identifiable because one or more of the design principles requires them to exist. They are not visible because they are not part of our current vision of the system. These “missing” centers contribute to the lack of wholeness and completeness in the system.

5. We choose one of these latent or missing centers to work on. It may be large or small, but we try to select one that seems the “most missing.” To be identifiable in this way, we must have an understanding of what needs to go there, either as a specific, well-defined entity or as an abstraction that captures what we know about the missing element.

6. We use the transformational aspect of one or more of the design principles, in particular those principles that highlighted the latent center, to differentiate and strengthen the latent element and the system as a whole.

7. As a result of the differentiation that occurred, new centers (latent and visible) will come into being. These new centers strengthen the presence of the design principles within the structure of the system.
8. Another result of this differentiation is that larger centers, those which contain and are related to the latent center we chose to work on, are strengthened. We have also strengthened centers that are parallel to these larger centers as well as centers that fill smaller supporting roles in the structure of the system. As a whole, the system will now be stronger, more complete, and more coherent.

9. We test the system to make sure that it actually is stronger, more complete, and more coherent, and that we have moved closer to solving the overarching problem. If this is not the case, we revert the system back to our starting configuration before this differentiation and repeat the process, either choosing another center to work on or using other principle(s) as a basis for differentiation.

10. We also test that what we have done is the simplest differentiation possible with respect to our goal and the center we chose to work on. If this is not the case, we repeat the process, working on the targeted center from this iteration and applying at least the Simplicity and Inner Calm principle.

11. When complete, we go back to the beginning of the cycle and apply the same process again. We continue to repeat the process until we can no longer strengthen the system and it is a complete solution to the initial problem.

   It is important to note that this process, either as Alexander states it or in our interpretation, is a recursive process. The key point where recursion occurs is after the choice of the center to work on. We may work with that center in its own configuration within the whole, or we may choose to consider that center as a whole in its own right and re-initialize the process at that point. In the second case, we must have an understanding of the design problem that exists at this center’s level in the overall configuration.

3.3.4 The Decision Pattern

Our initial motivation for the development of the Decision Pattern was to have some means for the students in the pilot study to document their design processes other than a simple narrative which we expected to be vague and incomplete. As the pilot study was conducted as part of a regular course and the study’s design exercises would also be graded, we wanted a way to evaluate how the students arrived at their solutions. From a teaching and learning perspective, we also wanted to stimulate the students’ critical thinking skills[128]. Additionally, we recognized that such a design documentation tool or structure should be an integral part of the solution development process and should not be more time-consuming or important to the student than solving the problem itself. We considered several alternatives before developing our own documentation structure.

One alternative we considered is the argument map. Argument maps are graphical representations that provide a strict convention for diagramming sequences of logical arguments in support
of a chain of reasoning[247]. This sequence of arguments allows the instructor to quickly trace the thought processes that the student followed to arrive at the solution. Figure 3.21 illustrates a very simple argument mapping. While there was argument mapping software available at the time, we felt that learning to use the software would have introduced an additional layer of complexity to the design problems not directly related to the development of the solution.

"How much is your life worth to you?" is an idiotic question.

Reason:
No amount of money could compensate you for the loss of your life.

Reason:
The money would be no good to you if you were dead.

Figure 3.21: Simple argument map

Design rationale [184], a term identifying a range of approaches for capturing, recording, documenting, and effectively using decision rationales in the software design and development process, was another alternative considered for documenting design decisions. A simplified view of the conceptual core of design rationale is shown in Figure 3.22. A design starts with a set of requirements that define the system. These requirements are mapped to goals (and sub-goals if necessary) which may be satisfied by one or more alternatives. Each alternative is mapped to an artifact and/or a new requirement that is a consequence of that alternative. The designer must make a choice between the alternatives, and the rationale for that choice is represented argumentatively, as claims for and against each alternative.

There are a variety of different approaches to using design rationale[40, 62, 164, 171]. A common aspect of all is that the rationale is collected after the fact, through a review of video recordings, documents, and other artifacts produced in the development of a system[168]. Design rationale, as a software development tool, is primarily targeted at capturing organizational knowledge for future reuse, and was not well suited for our purposes, either as a classroom learning tool or as a research tool for the pilot study. However, the underlying premise, making
design choices by explicitly considering arguments for and against each alternative resonated with the “forces” and “counterforces” elements of design patterns.

Lowy and Hood assert that opposition and tension are axiomatic principles of mastering $2 \times 2$ modeling and problem solving[169]. Tension is the result of unresolved opposition within a situation or problem and provides the drive and focus needed to find a successful resolution. Opposition defines the problem and is the impulse driving towards a resolution. A simple (and commonly used) example of a $2 \times 2$ matrix is shown in Figure 3.23, which outlines the opposing forces in Pascal’s Wager [199, §233]. Resolving this dilemma requires making a decision under uncertainty and at risk, both characteristics of design decisions made in the development of software systems.

Selecting the dialectical conflict at the core of the problem and correctly identifying the opposing forces is critical to building an effective $2 \times 2$ model, and one of the most difficult aspects of using this problem-solving method[169, 170]. Using $2 \times 2$ matrix modeling also requires an understanding of contextual layers or hierarchies because as one conflict or tension is resolved, new ones are triggered at different logical levels in the overall situation. We concluded that the $2 \times 2$ matrix approach requires a high level of expertise to use effectively even though it is an elegantly simple representation of the forces that create a problem. The representation is simple, but the construction is often difficult. Furthermore, its apparent simplicity could be a stumbling block for novice designers because it is very easy to use incorrectly, which would waste time and effort[169].

Another familiar concept commonly taught in introductory programming classes is the Design-by-Contract structure for method or function specification[181, p. 331]. This structure requires documenting functions with specific preconditions and postconditions that precisely
describe the functions behavior. Preconditions are those conditions in the program that must be true before the function is called for it to successfully complete its computation. Postconditions are the properties of the program that will be true when the function execution is complete, including all return values and/or side effects.

As simple and straightforward as the design-by-contract idea is, it is intended as a code-level structure, and we felt that it was not flexible enough to handle the kinds of design decisions that students were expected to make at the higher levels of software system design. Also, it does not provide an adequate structure for describing why a particular decision is made, or the constraints and forces that make that decision the most appropriate resolution.

In a discussion of social influences on software architecture, Cockburn employs a four-element structure (intent, context, forces, and resolution) for documenting design decisions in a case study[59]. This structure provided most of the basic elements needed to capture how and why students made the decisions that led to their solutions. Because it is a compact template, intended to capture a single decision, it was hoped that students would be more receptive to documenting their work towards a solution during the process rather than after the fact. The template was presented as the solution element of the pattern shown in Figure 3.24.

There are two significant differences between this pattern and Cockburn’s design decision documentation structure. First, the Forces section explicitly requires listing counterforces. This was done to push students to actively consider both sides of each piece of information, e.g., the positive and negative influences of that force on the problem. Second, a Predecessor section was added to the pattern to capture the ordering and dependency relationships in a set of design
Pattern Name: Decision

Problem: How can novice software designers capture and communicate their motivation and justification for decisions they make.

Context: Open-ended software system design tasks incorporated into homework assignments and in-class exercises.

Forces:

- Narrative descriptions can be vague.
- Templates provide cues for necessary information.
- Documenting individual decisions supports targeted feedback at problem points.
- Structured documentation supports self-reflection over multiple decisions.

Solution: Use the following template to record design decisions:

Decision number - Chronological ordering of decisions.

Intent - The reason why this decision must be made.

Context - The question or problem this decision answers or resolves, its connection to the world/domain, and why it is important.

Forces and Counterforces - Information that defines the problem and limits the solution. Whenever possible, both sides (pro and con) should be given.

Resolution - How the problem is resolved, taking into account all of the forces and their relationships to and within the context of the problem.

Predecessor(s) - The decision(s) made prior to this one that resulted in the need to make this decision.

Figure 3.24: The Decision Pattern

decisions. This information was not necessary for Cockburn’s analysis. However, to assess the processes students use to solve design problems, this information identifies sequences of design decisions. As researchers and as teachers, we hoped that this would help us understand how earlier decisions influence later ones by studying decisions as parts of a sequence rather than as isolated events.

We can safely assume that every teacher has, at some time in their career, wondered why a student made a certain decision. We often ask this question when we suspect that their choice was more of a guess than a justifiable response, or because the choice did not fit with the other decisions that had been made towards completing the task or problem. The usual responses from students to this questioning are statements like “I saw a similar example in the textbook
As van Gelder noted, requiring students to use a strict format for documenting their decisions can provide insights into how the student arrived at a particular solution. This is similar to showing each step in a mathematical proof or derivation: a step represents the application of some operation or transformation intended to move the problem closer to a successful resolution. The teacher can pinpoint errors in judgement more easily and precisely, and provide feedback that addresses the specific mistake. Such direct and targeted feedback helps the student learn from their mistakes more effectively than general comments (e.g., often the teacher's or grader's best guess as to where the student's reasoning toward the solution went off track) about the incorrectness of a response or problem solution.

The **Decision Pattern** abstracts and structures the key elements necessary to make a critically-informed decision. It also supports student learning by providing a template for documenting their progress towards completing difficult problem-solving and design tasks. By effectively communicating these thought processes to the instructor, feedback to the student can address specific errors in judgement or logic that resulted in incorrect or inappropriate design artifacts, enhancing the instructor's ability to use written assignments as learning tools as well as assessment instruments.

### 3.3.5 Using the $\mathcal{P}^3F$

An example of how the $\mathcal{P}^3F$ is used will help the reader understand the framework and will illustrate how it was introduced to students in both the pilot study and the current research. The $\mathcal{P}^3F$ Differentiating Process is applied iteratively, with each cycle incorporating a design decision documented using the Decision Pattern. The problem is given as follows:

A local municipal Public Works Department has decided to develop a Web-based pothole tracking and repair system (PHTRS) to replace their existing telephone-based reporting system and manual documentation and tracking system. The town’s Request For Proposals provides the following description of the desired system:

*Citizens can log into a web site and report the location and severity of potholes. As potholes are reported, they are logged within a “public works*
department repair system" (PWDRS) and are assigned an identifying number, stored by street address, size (on a scale of 1 to 10), location (middle, curb, etc.), district (determined from street address), and repair priority (determined from the size of the pothole). Work order data are associated with each pothole and include pothole location and size, repair crew identifying number, number of people on the repair crew, equipment assigned, hours applied to repair, hole status (work in progress, repaired, temporary repair, not repaired), amount of filler material used, and cost of repair (computed from hours applied, number of people, material, and equipment used). Finally, a damage file is created to hold information about reported damage due to the pothole and includes the citizen’s name, address, phone number, type of damage, and dollar amount of damage.

In addition to allowing citizens to report potholes, they should also be able (using the citizen access portal) to search for and view the status of a previously-reported pothole. Public Works administrators should also be able to log into the PHTRS to view reports listing reported potholes and damage information. PHTRS is a Web-based system; all queries are to be made interactively. Writers of accepted proposals will be given more detailed information about the interface to the existing PWDRS to facilitate a more detailed system design.

For this example, the $P^3F$ process will be iterated five times with the designer’s thinking explained for each of the eleven steps in the process. The thinking presented here is based on a fusion of examples I demonstrated in various presentations of the $P^3F$. The explanations will be supplemented with drawings and other visual/graphical elements. Each iteration will also reference an instance of the Decision Pattern that documents the design decision central to that iteration.

Iteration 1:

1. The new system does not currently exist in any form other than the Request for Proposals document. However, key elements (centers) are described in this document: citizens, a web site, the PWDRS, potholes with descriptive information, work orders and related information, a damage file, and Public Works administrators. The desired solution will be a web-based application that provides a simple interface to the PWDRS that can be used by the general public to report potholes and obtain status information about their reports. Citizens will also be able to report damages caused by potholes. Public Works administrators will also be able to access information about reported potholes and damages.

2. The problem description implies that the PWDRS is an existing software system used to manage work performed by the Public Works Department and that it is responsible for creating and maintaining work orders. The PHTRS will interface with the PWRDS to create new work orders for citizen-reported potholes and to
access status information on previously-reported potholes. Citizens will use the web-based PHTRS to enter this pothole information. The PHTRS will also create and maintain damage files. Public Works administrators will also use the PHTRS to access information about potholes in the town. Figure 3.25(a) shows a sketch capturing these centers and their connections.

3. There are several significant weaknesses in what we know about the system as a whole. Phrased as questions, some of these weaknesses are: How will citizens and administrators interact with the system? How will the PHTRS interact with the existing PWDRS? What information will a citizen need to enter to report a pothole? Where does the additional information in the work order record originate? Where is pothole data stored? Where are damage files stored? These questions are all related to the boundaries between the new pothole system and the entities that need to interact with it.

4. These boundaries are latent centers: we need well-defined boundaries between the new system and the interacting entities, but they barely exist at this time.

5. The boundary between the citizen and the new system is a critical part of the system and it is only vaguely defined at this time. Since the PHTRS is intended to track information about potholes and the citizen interface is where this information will be entered, it is the “most missing” element, and a latent center that should be strengthened.

6. There are two parts to citizen portal interface, based on the given problem description: entering pothole information and searching for the status of previously entered potholes. This differentiation is guided by several principles: Good Shape requires each element to be fit and specific to its purpose; The ability to put data into the system and to retrieve it later is an instance of Local Symmetry; Pothole information is the Deep Interlock that helps unify the system while maintaining Ambiguity by hiding the storage and retrieval mechanisms from the users.

7. As a result of this differentiation, the boundary between citizens and the PHTRS emerges as a system of at least two new centers as shown in Figure 3.25(b).

Figure 3.25: Using the $\mathcal{P}^3\mathcal{F}$: Iteration 1 artifacts
8. More is now known about the system as a whole, particularly the relationship between the center representing the citizen users and the one representing the core of the PHTRS.

9. This step has moved the system closer to a solution to the problem as a whole.

10. Based on the information at hand and the chosen center of interest, this differentiation was the simplest possible.

11. The solution is not complete, so another iteration will be performed. The decision made in this iteration is documented using the Decision Pattern in Figure 3.26.

| Number: | 1 |
| Intent: | To refine the boundary between the citizen and the PHTRS. |
| Context: | The purpose of the new system is to allow citizens to report potholes using a web-based interface and to be able to search for previously reported potholes. This is the first step in developing a solution that meets the requirements given in the customer’s problem statement. |
| Forces: | a) Information systems must get data from somewhere; b) if you cannot get information out of the system, it is not very useful; c) details of data manipulation should be hidden from users; and d) users should be able to clearly distinguish between entering data into the system and getting it back out. |
| Counterforces: | a) Users need a single point to access the system; b) different user interfaces mean more code to write and maintain; and c) users of web-based systems are smart enough to know how to use the system. |
| Resolution: | The boundary between citizen users and the PHTRS is the “Citizen Portal” and contains at least two centers: one for a data entry interface and one for a search interface. |
| Predecessor(s): | None |

Figure 3.26: Using the \( P^3F \): Decision 1 documentation

Iteration 2:

1. The current state of the partially developed system is shown in Figure 3.27(a).

2. Some order and organization has been added, but there is still much missing. In particular, the Levels Of Scale suggests that there should be parallel centers between the PHTRS and the other system entities that are closely related to it.

3. These other four boundaries are critical to defining what the PHTRS can and cannot do in its interactions with users (citizens and PW administrators) as well as with the existing PWDRS.

4. The systems of centers that make up these boundaries are latent: they need to be there and to be well-defined, but they do not yet exist. They can be sketched in as temporary placeholders as in Figure 3.27(b).
5. Details of the boundary with the PWDRS are not included in the problem statement, so that can be deferred until more is known about the new system. The “damage files” and “potholes” boundaries seem to be dependent upon what the human users of the system do, so these boundaries can also be temporarily deferred. That leaves the boundary between the PHTRS and the Public Works administrators as the current center of interest.

6. The problem statement states that Public Works administrators will be able to “view reports” of two basic types, potholes and damage information. The need for these two reports implies that these are important kinds of information *Strong Centers* within the system and should be differentiated in the *Boundary* between the administrators and the PHTRS.

7. As a result of this differentiation, the boundary between the Public Works Administrators and the PHTRS emerges as a system of at least two new centers as shown in Figure 3.27(c).

8. The overall order of the system is stronger as a result of defining the relationship between the center representing the Public Works administrator users and the one representing the core of the PHTRS.

9. This step has moved the system closer to a solution to the problem as a whole.

10. Based on the information at hand and the chosen center of interest, this differentiation was the simplest possible.

11. The solution is not complete, so another iteration will be performed. The decision made in this iteration is documented using the *Decision Pattern* in Figure 3.28.
### Number: 2

#### Intent:
To refine the boundary between the Public Works administrators and the PHTRS.

#### Context:
System design has started and to maintain parallel *levels of scale* the boundaries between the system and other external entities need to be developed further.

#### Forces:
- **a)** Administrators are interested in viewing reports of reported potholes and damages to vehicles;
- **b)** these two reports involve different sets of information and should be handled separately; and
- **c)** administrators and citizens are two different types of users and should have different interfaces to clearly distinguish them.

#### Counterforces:
- **a)** Reports are reports;
- **b)** administrators may be citizens, too; and
- **c)** more interfaces means more code to write, text, and maintain.

#### Resolution:
The boundary between administrator users and the PHTRS is the “Administrator Portal” and contains at least two centers: one requesting a report listing potholes and the other requesting a report listing vehicle damage information.

#### Predecessor(s): 1

---

**Figure 3.28:** Using the $P^3F$: Decision 2 documentation

---

### Iteration 3:

1. The current state of the partially developed system is shown in Figure 3.29(a).

2. The latent boundary centers that remain around the PHTRS are those with the damage files, the potholes, and PWDRS. These boundaries must be defined and strengthened to increase the wholeness and order of the system design.

3. The weakness of the PHTRS – PWDRS boundary is becoming more critical to understanding how the system will work. While the details of this interface will not be completely known until the proposal is accepted, there needs to be at least an abstract definition of this boundary to continue developing the system.

4. The problem statement indicates that reported potholes will be logged in the PWDRS by the PHTRS. The *Void* principle suggests additional latent centers exist in the PHTRS and in the PWDRS.

5. The latent pothole report and pothole centers in the PHTRS and PWDRS are needed to contain pothole reports and potholes, respectively.

6. Using the *Void* principle transformation, a new center representing a pothole report is added as a nested center in the PHTRS. Using the same transformation, a new center is nested in the PWDRS to represent the information about a pothole.

7. The *Not-Separateness* principle suggests that these latent centers would refer to actual potholes that are reported by citizens, so that “potholes” as physical entities would be better associated with citizens rather than the PHTRS. Also, this differentiation results in the emergence of a new latent center within the “work order” center. This center should represent a work order corresponding to a reported pothole. The new view of the system resulting from this differentiation is shown in Figure 3.29(b).
8. The centers representing the PHTRS and the PWDRS have been strengthened by this decision, as they are better defined within the system as a whole. Also, the citizen and administrator centers have been strengthened because the flow of information through the system is clearer.

9. The system is stronger, more coherent, and closer to achieving the overall goals than it was before this decision.

10. This differentiation was kept simple, deferring additional details until a later iteration when the system as a whole is better understood.

11. The solution is not complete, so another iteration will be performed. The decision made in this iteration is documented using the Decision Pattern in Figure 3.30.

**Iteration 4:**

1. The current state of the system is shown in Figure 3.29 (b).

2. The system is becoming more organized and more understood. The details of the PHTRS – PWDRS boundary and the PHTRS – damage files boundary are still not known.

3. While the previous iteration strengthened the PHTRS – PWDRS boundary through the emergence of information-carrying centers in both systems, this interface still remains critical to the development of the system.
Number: 3

Intent: To refine the boundary between the PHTRS and the PWDRS.

Context: Potholes are reported by citizens through the PHTRS, which “logs” the reports in the PWDRS in order to create a work order to repair the pothole. Nothing in the current view of the system represents this information flow.

Forces: a) An abstraction of the information flow needs to be defined to support the development of other parts of the system; b) The PHTRS is intended to allow citizens to report potholes; and c) The PHTRS logs pothole reports into the repair system (PWDRS).

Counterforces: a) Details of the PHTRS – PWDRS interface are incomplete; b) The PHTRS may not need to store pothole information; and c) The PWDRS may not be modifiable.

Resolution: Abstract entities are added within the PHTRS and the PWDRS to represent pothole reports and pothole repair information, respectively.

Predecessor(s): 1, 2

Figure 3.30: Using the P3F: Decision 3 documentation

4. Because it is still void and undefined, the structure of the PHTRS – PWDRS boundary center is still latent and contributes to the lack of wholeness and coherence within the system. The boundary between the PHTRS and the damage files is also latent, but is not as critical to the development of the system as a whole at this point in time.

5. Because of its criticality, the PHTRS – PWDRS boundary center will be the focus of attention for this iteration. The problem statement indicates that pothole reports are “logged within the PWDRS” and that work order data are associated with each pothole. This suggests that the PHTRS will use the PWDRS to programmatically create a work order to repair a reported pothole. The problem also specifies that users will be able to view the status of previously-reported potholes, which would seem to require accessing parts of the work order data.

6. The PHTRS – PWDRS boundary is differentiated based on The Void, Local Symmetry, and Contrast principles.

7. This differentiation generates two nested centers representing the functionality this interface must provide: creating a work order for a reported pothole and checking the status of a previously-reported pothole, as illustrated in Figure 3.31(a).

8. The new PHTRS and the existing PWDRS are both strengthened by this differentiation and transformation of the latent boundary between them. The new configuration of the system is shown in Figure 3.31(b).

9. The system is stronger, more coherent, and closer to achieving the overall goals than it was before this decision.

10. Based on the information at hand and the chosen center of interest, this differentiation was the simplest possible.

11. The solution is not complete, so another iteration will be performed. The decision made in this iteration is documented using the Decision Pattern in Figure 3.32.
**Iteration 5:**

1. The current state of the system is shown in Figure 3.31 (b).
2. The wholeness of the system is reflected in the organization of the strong centers identified and developed in previous iterations.
3. The most visible weakness at this point in time is the relationship between the PHTRS and the damage files. Also, the current design does not incorporate any means to enter damage information into the system.
4. The centers that would define this relationship and provide the specified functionality are latent and need to be defined and strengthened.
5. Since these centers are very closely related, they will be worked on together. The latent damage file center represents where vehicle damage information will be stored and the latent data entry functionality in the citizen interface provides the means to get the information to be stored.
6. The Void transformation is applied, generating a new center within the PHTRS to represent the storage of damage file information. Individual damage reports are associated with previously reported potholes. A new center is necessary in the citizen portal interface to provide the Deep Interlock and Ambiguity that allows damages to be entered into the system. Because the damage file center has been moved to a nested position within the PHTRS, the Positive Space principle requires removing the unnecessary external center and its latent boundary with the PHTRS. The changes due to this differentiation of the Citizen – PHTRS boundary are shown in Figure 3.33(a).
7. New centers representing damage files and damage reporting have been added to the system. Additional centers that connect these new entities to each other and to the
Number: 4

Intent: To refine the interface between the existing PWDRS and the new PHTRS.

Context: Potholes are reported by citizens using the PHTRS, which then must use the PWDRS to associate this information with a work order to repair the street. Users should also be able to look up a pothole that has been reported to find out the status of the repairs. Details about the programmatic interface to the PWDRS are not known at this time, but the basic functionalities of creating and searching for a work order are assumed, abstractly, since these are core operations of this system.

Forces: a) Citizens want potholes fixed promptly; b) street repairs are made by the Public Works department based on work orders entered in the PWDRS; and c) citizens may want to find out when a pothole they reported will be repaired.

Counterforces: a) Details about the PWDRS are not given in the problem statement; b) making decisions about this interface without all of the details could cause problems later in the design and development process; and c) Other parts of the system might be more important to work on at this time.

Resolution: Create an abstract interface based on simple functionality that is assumed to be necessary to the basic operation of the PWDRS: creating and looking up work orders.

Predecessor(s): 3

Figure 3.32: Using the $P^3F$: Decision 4 documentation

citizen user have also been added, although the structure of these connections is still latent at this time.

8. The PHTRS center is strengthened through a clearer definition of the information it must manage directly. The Citizen Portal center (Citizen – PHTRS boundary) has also been strengthened and now provides the functionality required in the problem statement. Figure 3.33(b) shows the current view of the system as a whole.

9. The system is stronger and closer to meeting the goals as a result of this differentiation. The problem statement specifies that citizens will be able to report damage to their vehicles cause by potholes, and this iteration provides that ability in an abstract way.

10. Based on the information at hand and the chosen center of interest, this differentiation was the simplest possible. Additionally, the system is now simpler overall because an external entity has been integrated into the system itself.

11. The solution is not complete, so another iteration will be performed. The decision made in this iteration is documented using the Decision Pattern in Figure 3.34.

To the experienced designer, this sequence of steps may seem obvious or trivial, and hardly worth the effort expended to document these decisions. However, this sequence of design decisions is not typical of those made by computer science students who have worked on this problem before being introduced to the $P^3F$. Instead of establishing the basic relationships between the entities described in the problem statement, they generally jump right to “needing” an Apache web server, a MySQL database server, and a Linux-based file server. Figure 3.35
illustrates the typical approach to solving the pothole tracking problem taken by many observed novice software designers.

When asked to explain their design, the responses are usually general: “Web pages are used for getting data from users and displaying reports or search results;” “Customers want low-cost solutions so I’m using open-source software for everything but the program I’m writing;” “I think that the PWDRS will just become part of the new system I will write.” Novices rarely consider users of the system until they are close to writing code, if at all, and will usually choose whatever hardware and software architectures they are familiar with, ignoring the customer’s existing investments in these resources.

3.4 Summary

This chapter began by summarizing the weaknesses novice designers exhibit and the characteristics of a support framework to help them behave more like expert designers. In particular, novices tend to take a restricted and implementation-oriented view of design problems, which is a critical factor in many of the other novice weaknesses. The need for a set of general purpose abstractions capable of describing relationships and interactions within a complex system was also identified. The framework should also incorporate fundamental design principles as a basis for generating alternatives and making design decisions. Noting that students/novices have a preference for problem-solving “recipes,” the support framework should employ a step-by-step process to help guide the designer’s thinking and decision-making as they work through a
<table>
<thead>
<tr>
<th>Number:</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intent:</strong></td>
<td>To refine the boundary between the PHTRS and the damage files.</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>Citizens need to be able to enter pothole-cause vehicle damages into the PHTRS, which needs to maintain this information for administrative purposes. The relationship between damage files and the PHTRS is the only high-level interface that has not been defined to any extent.</td>
</tr>
<tr>
<td><strong>Forces:</strong></td>
<td>a) Citizens want to report potholes so they can be reimbursed for damages caused by unrepaired potholes; b) storing damage data in the PHTRS minimizes the risks associated with managing external data files; c) storing damage data in the PHTRS simplifies reporting damages to public works administrators; and d) eliminating external interfaces makes the system simpler.</td>
</tr>
<tr>
<td><strong>Counterforces:</strong></td>
<td>a) The town really does not want to reimburse people for pothole damage; that is what they are supposed to have insurance for; b) each damage instance has a whole “file” of information that may be associated with it, such as photos, repair estimates, the kind of car involved, and a pothole identifier; and c) some of the information may not be received in an electronic form.</td>
</tr>
<tr>
<td><strong>Resolution:</strong></td>
<td>Logically nest a center representing damage file data inside the PHTRS and add an element to the citizen portal to allow damage reporting. Additional refinement of the damage file center will be made based on information requests from the Public Works Department.</td>
</tr>
<tr>
<td><strong>Predecessor(s):</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 3.34: Using the \( P^3F \): Decision 5 documentation**

problem. The Principles, Patterns, and Process Framework was designed to meet these goals.

The fifteen core principles in the \( P^3F \) provide a foundation for the novice’s design process within the framework. Expressed as visual and geometric metaphors, their abstract nature is buffered by relating them to naturally-occurring and human-built structures in the real world, examples that novices can physically see and touch to bridge the concrete and abstract. Extending their exposure to design patterns, these fifteen principles are presented in a pattern structure. This structure allows each principle to be compactly associated with the forces and counterforces they balance, observational examples of the value the principle imparts to a system, a discussion of the transformational application of the principle, and a list of closely related and/or interdependent principles.

The \( P^3F \) accommodates the desire for a “design recipe” with an adapted fundamental differentiating and unfolding process. To encourage novice designers to create their own “big picture” view of the design problem and solution, the first steps of the process refer directly to the whole system as it is situated in its operational environment. The last steps of the process also refer the designer to the whole system through a reflective look back on the decision that was just made and how it affects the system. The Decision Pattern embodies this process by asking the designer to specifically identify the intent and context of the design decision, the competing forces balanced and resolved by the decision, and a description of the resolution of the problem. It also provides a mechanism for ordering decisions to support tracing sequences
of decisions and reflecting upon what the designer has done and why.

It is important to note that the $P^3F$ is not yet another software design or development methodology. Design methods, at their core, follow some form of the basic model shown in Figure 3.36: a problem is identified, the design method is applied to the problem, and a solution is generated. There are many software design and software engineering methodologies, waterfall[214], structured programming[139], spiral[30], opportunistic[47], and eXtreme Programming[27], to name a few. These methodologies are intended to guide the large-scale process of developing software systems by structuring the flow through the various phases of the process.

The $P^3F$ differs from these methodologies in that it does not try to manage the software
development process. Instead, the framework is a guide to thinking about a problem and making reliable and traceable decisions towards designing a solution. The $P^3F$ is designed to be a guide to thinking about design problems, as illustrated in Figure 3.37. As a thinking tool, the framework helps the novice view a design problem in context and develop an understanding of the different perspectives and influences that affect the problem. Through this understanding, the designer is able to see many possible solutions, many of which are not appropriate for the problem in its context but nonetheless valuable to acknowledge.

![Figure 3.37: Design Thinking: What the $P^3F$ is](image)

The $P^3F$ is naturally recursive as a result of the definitions of the fundamental design principles, and this recursive nature extends to the definition of a design problem in context. This provides the framework with flexibility and adaptability, allowing it to be useful in any phase of software system development. During the initial phase of identifying the problem and eliciting requirements, the $P^3F$ can help a novice relate the various perspectives on the overall problem(s) that a system will be expected to resolve. In the design and implementation phases, the framework helps to maintain a “big picture” view of the system that incorporates the relationships of all of the system components as well as the interactions of the system with external entities that are part of the larger system in which the software must operate.
Chapter 4

Research Methodology

The purpose of this chapter is to describe and justify the research methodology and study design developed for this research project. This justification is of particular importance due to the scarcity of research in this area. The research methods chosen must be capable of addressing the issues central to the research problem, specifically the eight research questions and objectives listed in Chapter 1. The previous two chapters have addressed the first four research questions through an analysis and synthesis of the existing literature. This analysis identified several key strategies used by expert designers and critical weaknesses novice designers use to approach complex, open-ended system design problems. Specific expert strategies were mapped to novice weaknesses and the results used to guide the development of a design learning framework, the Principles, Patterns, and Process Framework. The next step in this research is to experimentally evaluate the $P^3F$ through the lens of the remaining research questions:

1. Are software design students able to understand the basic structure and application of the framework and the decision-making process it incorporates?

2. Are software design students able to demonstrate the expert strategies incorporated into the framework when presented with a complex, ill-structured design problem?

3. Does the framework improve the quality of design artifacts produced by the student?

4. Does the framework improve the student’s perception of his or her design abilities and enable them to be more confident in their design decision-making?

The first of these questions addresses the structure and organization of the $P^3F$ and if the target audience, advanced undergraduate software design students, can actually understand it. The other three questions seek to identify and describe the effects of introducing the framework. Taken together, these four questions provide four different views of the effects of the $P^3F$ on the
research subjects, contributing to a broader understanding of these effects. Section 4.1 of this chapter examines the research questions in greater detail to establish what research methods will be most applicable for answering each research question, followed by a discussion of the study design and implementation in Section 4.2.

### 4.1 Research Methodology

The choice of methodology for a research study is driven by the nature of the research questions and the criteria established for successfully answering them[71]. Table 4.1 presents some possible success criteria for each of the four research questions listed above. These criteria serve as a starting point for evaluating potential research methods and study designs in the discussion that follows.

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Success Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Able to understand the $P^3F$</td>
<td>Correctly use $P^3F$ principles when solving a design problem</td>
</tr>
<tr>
<td>2. Demonstrate expert strategies</td>
<td>Correctly use $P^3F$ process when solving a design problem</td>
</tr>
<tr>
<td></td>
<td>Correctly use the Decision Pattern when solving a design problem</td>
</tr>
<tr>
<td></td>
<td>Maintain big-picture view of design problem and solution</td>
</tr>
<tr>
<td></td>
<td>Use abstractions at different levels to organize problem/solution information</td>
</tr>
<tr>
<td></td>
<td>Navigate the problem/solution space opportunistically</td>
</tr>
<tr>
<td></td>
<td>Make design decisions confidently</td>
</tr>
<tr>
<td></td>
<td>Justify design decisions</td>
</tr>
<tr>
<td>3. Improve quality of design artifacts</td>
<td>Solutions meet given requirements &amp; goals</td>
</tr>
<tr>
<td></td>
<td>Solutions are feasible with respect to given resources</td>
</tr>
<tr>
<td></td>
<td>Solutions are realistic with respect to expected number of users, data volume, security, and similar constraints</td>
</tr>
<tr>
<td>4. Improve perception of own design abilities</td>
<td>Associate expert behaviors with their own design thinking</td>
</tr>
<tr>
<td></td>
<td>Feel more confident about design decision-making and ability to justify their decisions</td>
</tr>
<tr>
<td></td>
<td>Willing to ask questions and/or seek new information to solve problems</td>
</tr>
</tbody>
</table>

A first step in this discussion is to characterize each of the research questions in order to determine what research method(s) would be most applicable to the question and the success criteria. Research questions 1 and 2 involve examining the behavior of the research subjects
to identify changes in behavior after being introduced to the $P^3F$. Specifically, they require that behavioral characteristics of the subjects as they work must be identified for use as a data source. Both of these questions also suggest that the subjects will perform design-related tasks and that some kind of record of their work will be captured and analyzed in order to identify and measure aspects of their design behaviors.

The evaluation of design artifacts in research question 3 requires that the research subjects produce artifacts representing their solutions to a design problem. This question and its success criteria also suggest that some kind of “measure of quality” will be defined and used to objectively evaluate subject-produced artifacts. It also requires that the design problems are constructed such that the subjects are able to produce various drawings, diagrams, text, source code, and/or other forms of documentation and implementation of their solutions, and that these solution artifacts can be collected for evaluation.

Research question 4 involves an examination of the subjects’ thoughts about their own design skills and strategies. While these self-perceptions might manifest themselves as a subject works on a design problem, it is doubtful that this would be a reliable method for collecting consistent data about how the subjects view their design skills. The overarching goal of this question is to associate the introduction of the $P^3F$ with the research subjects’ perception of their own design skills, with the expectation that they will become more confident in their abilities and will adopt more expert-like attitudes about designing software systems. This goal suggests the use of a consistent instrument for characterizing these attitudes and self-perceptions.

In order to meet the overarching goal of this research, three conditions must be present: (1) the $P^3F$ must initiate an observable change in the way novice software designers approach design-related problems; (2) the $P^3F$ must invoke a change in novices’ attitudes and self-confidence about how they make design decisions; and (3) the $P^3F$ must cause an improvement in the quality of the designed artifacts produced by novice software designers. The convergence of positive results for these three conditions would thus corroborate the hypothesis that the $P^3F$ is an effective tool for helping software design students learn and use expert design strategies. These three different conditions suggest the use of a multi-modal research approach that will employ different types of data collection and analysis to confirm or refute each of the conditions.

This research lends itself well to the use of a interrupted time series quasi-experiment as the experimental method[42]. Before and after (or one-group pretest-posttest) studies are often employed to identify improvements in student learning as a result of using particular learning tools[28]. In this type of experimental design, the subjects serve as their own control group. This helps to minimize threats to validity caused by differences between the control and treatment groups, particularly when evaluating learning and/or problem-solving behaviors[61, 224].

An interrupted time series design extends this experimental method by adding additional observations before and after the treatment intervention. The added observations allow better
estimates of the impact of the treatment. When a change in slope or intercept occurs immediately following the treatment, causal inferences are subject to relatively few threats to internal validity[225]. The most important threat to the internal validity of time-series experiments is that of history: some other event occurs in close temporal proximity to the treatment event and this other event also produces the effect that is hypothesized from the treatment. The collection of different types of data will help to significantly reduce this threat. The remainder of this section considers the alternatives available for collecting the three kinds of data needed to answer the research questions.

4.1.1 Methods for Observing Behavior

Observation-based research methods can be classified into three categories, based on the level of interaction the researcher has with the subjects. In a direct observation study the researcher strives to remain as unobtrusive as possible, concentrating on recording what happens without taking an active part in the observed activity. Studies using the participant observation method allow the researcher to take part in the activity while observing and recording events as they occur. Action research involves the researcher as a complete participant in the activity with the intention of directly influencing the events that occur. Each of these research methods has advantages and disadvantages depending upon the goals of the research study[71, 212].

Direct observation has the advantage that the observation work is wholly dedicated to gathering data. However, the researcher/observer is one step removed from the activity under study and may miss or incorrectly interpret key events that occur, resulting in incomplete data. To address this problem, direct observations are often supported through other research methods such as interviews or surveys. This research method is best suited for situations where the actions or events of interest are well-defined, for example, recording the time spent performing specific tasks within a larger process.

A major advantage of participant observation is that it allows the researcher to explore more subtle aspects of the activity under study. The observer does not seek to influence the subject’s actions, but may ask questions or make comments to elicit details about what the subject is doing. One significant disadvantage of this method is that it divides the time and attention of the observer, potentially omitting important data. Because the researcher does interact with the research subjects, this research method is also more likely to be affected by bias introduced by the researcher’s interactions.

Because the researcher is an active participant with the intention of directly influencing the subjects behavior, action research allows the observer to experiment with new approaches and test hypotheses that may not have been known before the study began. The advantage of action research is that it can be dynamically responsive to the progress of the study, allowing
the researcher to immediately respond to subjects’ feedback. This flexibility comes with a higher risk of observer bias. Also, because the results of the research are partially a product of the researcher’s involvement, action research studies do not generalize well.

Other options are common to all three general types of observational studies. A critical choice is whether or not the observer or technical instrument will be visible to the research subject. Subjects may behave differently if they know they are being observed, but in many cases hidden observation may not be possible. Hidden observation of human research subjects is generally an option only when the research is conducted in an environment where the subjects know that they are routinely observed, such as in a workplace environment. Otherwise, subjects must be informed that they will be observed in some fashion, even if the actual observer or instrument is hidden.

The rigidity of the observation structure is an important choice that must be made and is dependent upon the nature of the research. Explorative studies will tend to have less structure, incorporate more open-ended questions or scenarios, and allow the subject more choices of actions or activities. Studies investigating well-defined processes or finely focused research questions are generally highly structured.

Another key choice that must be made in the design of an observational study is the environment and context of the observed situation. A research scenario may be designed strictly for investigation or it may be a situation that would normally occur as part of the subject’s day-to-day activities. The observation situation may be situated in the context of professional practice or it may be an educational exercise.

Ambiguity and incompleteness are characteristics of complex and ill-structured problems[228]. Because this research investigates how novice designers approach these kinds of problems, they are expected to ask questions to clarify and refine the details of the problems. The kinds of questions asked and how they are ordered are important indicators of how a subject is reasoning about a problem. The interaction with the subjects should be restricted to acting as the “customer” or other significant stakeholder in the system and must not exert any influence on how the subjects think about and work on the problems. The information provided in response to subjects’ questions should be as consistent as possible throughout the study.

To overcome the risk of the observer missing parts of the subjects’ activities and to provide a reviewable record of the observations, the sessions will be audio/video recorded. Video recording is an appropriate choice for this study because it captures non-verbal activities such as writing, sketching, and typing in addition to the subject’s verbalizations, providing richer and more complete data for analysis than is possible with an audio recording. The recording equipment will not be hidden from the subjects, but will be placed in an unobtrusive position.

While a pilot study using elements of the $P^3F$ has been conducted, the current work is still exploratory and some aspects of the study will be open-ended with respect to the subjects’
responses. To enable comparative analysis, subjects should be given the same problems to solve, but there should not be pre-defined “correct” solutions, giving the subjects the freedom to explore the problems in their own ways. The complexity of the design problems and the presence of a “customer” will help to give the observation sessions more of a real-world, professional practice context rather than that of an educational exercise. Controlling the observational environment in a private office will limit external distractions and interruptions.

4.1.2 Methods for Self-Assessment of Design Skills

The final objective of this research is to investigate any changes in the subjects’ perceptions about their own design skills and strategies that correlate with their exposure to the $P^*F$. This will require some method of drawing out the subjects’ thoughts and feelings about how they approach and think about complex design problems and situations. Journals, interviews, and surveys are three common methods employed in this type of research study.

Journals are frequently used in long-term studies, particularly when the subject cannot be directly observed over the course of the study[232]. Ideally, journal entries are recorded in real time rather than retroactively. The major disadvantage of using journals is that they may not provide a complete record of the subjects’ thoughts and reflections. Details may be omitted or inaccurately documented, or they may be “backfilled” by recording their work in retrospect. Maintaining a reflective journal is a skill that must be learned and students are often resistant to adopting this method of recording their thoughts[112]. Additionally, since the subjects will only be performing study-related activities during the observation sessions, maintaining a full-time reflective journal would be an extra burden on the subjects.

Interviews could be conducted as part of the research study sessions, eliminating the need for the subjects to perform study-related tasks outside the formal sessions. Interviews offer the researcher the opportunity to explore subjects’ responses in greater detail through extended questioning and discussion. Interview transcripts and notes can be a rich data source as a result. However, there is also a risk of inconsistent results between subjects and across multiple interviews as a consequence of this extended questioning[133]. Also, some subjects may not be comfortable talking about how they feel about themselves.

Surveys are one of the most commonly used research methods and are frequently used in self-control studies to evaluate the effectiveness of the experimental intervention. Subjects are asked to complete pre-treatment and post-treatment surveys, providing a measurement of any change that occurs as a result of the treatment intervention. Surveys may contain any combination of open-ended or closed-response questions, guided by the objectives of the research. As with an interview, responses to open-ended questions can be difficult to code and analyze. Closed questions limit responses to a fixed set of answers, simplifying the analysis process.
Surveys investigating attitudes usually use a Likert scale ranging from “Strongly Agree” to “Strongly Disagree” with each response category given a different numerical weight (e.g., Strongly Agree = 5 and Strongly Disagree = 1).

The self-assessment instrument chosen must be effective and appropriate with respect to the objectives of the experiment. The particular goal to be addressed by this part of the research is to determine if the subjects’ attitudes and perceptions of their own design skills and strategies are changed after being introduced to the \( P^3 \). Using a reflective journal to collect this data would be very inefficient due to the training time required as well as the expectation of poor self-reporting because of the subjects’ lack of experience and supervision. Interviewing would be better, but could take a significant amount of time at each observation session. A survey-type instrument can be designed with questions based on the expert and novice strategies identified in the literature review and in the pilot study. This survey could be compact enough to be completed by the subjects in a manageable time period as part of the study sessions, and would facilitate the before-and-after analysis of the subjects’ thoughts about their own design processes.

4.2 Research Study Design and Implementation

One-group pretest-posttest studies are commonly used to identify changes that occur in a population as a result of some treatment or intervention, particularly in research involving learning and other cognitive processes[28]. Variables of interest are measured before and after the treatment and analyzed to determine what changes occurred and if the treatment was a causative factor in those changes. Differences between control and treatment groups can be a threat to the validity of research investigating learning or problem-solving behaviors. The one-group experimental design helps to minimize this threat by using the subjects as both control (pretest data) and treatment (posttest data) groups in the study.

The one-group pretest-posttest design is frequently extended through the addition of one or more tests or observations before and after the treatment. The added measurements provide a better estimate of the treatment’s effects on the subjects. Changes in the slope or intercept of the series of measurements immediately following the treatment allow causal inferences with relatively little threat to the internal validity of the experiment[225]. The most critical threat to internal validity remaining in this design is the occurrence of an external event in close proximity to the experimental intervention that could produce an effect similar to that which is hypothesized from the treatment. In this study, the collection of different types of data reduces this threat by looking for related but different kinds of treatment effects.

The interrupted time series quasi-experiment designed for this research study used two observation points before the introduction of the \( P^3 \) and two more observations after the treatment. The experiment was planned for eight to twelve subjects selected from undergraduate
and first-year graduate students with little or no professional software development experience as representative of novice software designers. The observations were performed at two to three week intervals, working around university holidays and the subjects’ personal schedules.

The observations consisted of work sessions where the subjects worked on solving complex, open-ended software system design problems adapted from problems and examples in a variety of software engineering and systems analysis/design textbooks. At each session, subjects were given two problems and allowed to chose the one they wanted to work on during that session. All four observation sessions were audio and video recorded and the students were encouraged to “think aloud” as they work on the design problems. Think aloud experiments are a rigorous empirical method used to model the cognitive processes that a participant engages in when performing a task[248]. Ericsson and Simon argued that the think aloud method is a valid and reliable means of eliciting data on thinking and found no evidence that sequences of thoughts changed when subjects think aloud compared to others who worked silently[97].

At the start of each observation session, subjects were asked to complete a questionnaire to assess their own design abilities and skills. The $P^3F$ was introduced immediately following the second pretest observation and consisted of a in-person presentation of the framework and its use, a review of $P^3F$ web site, and several iterations of the $P^3F$ process working through the problem that was not chosen for that session.

Primary data collected were self-assessment questionnaires completed by the subjects at each observation session, audio-video recordings of the problem-solving episodes, and the paper and/or electronic artifacts representing the subjects’ solutions to the given design problems. After completing all of the activities in the fourth session, subjects were asked to complete an exit survey to gather their feedback about the $P^3F$ and their participation in the research study. The remainder of this section describes the design and implementation of this research study in detail.

4.2.1 Research Tools and Instruments

This study required the design of several research tools to facilitate the activities completed by the research subjects in addition to the survey instruments used to measure the subjects’ attitudes about their own design skills and to gather feedback on the $P^3F$ and their experiences as participants in this research. The tools developed included practice problems to help the subjects become comfortable with verbalizing their thoughts while working, the software system design problems themselves, and a web-based presentation of the $P^3F$. The development of these tools and instruments are described in the following discussion.
4.2.1.1 Web-based $P^3\cal{F}$ Presentation

In the pilot study, the $P^3\cal{F}$ was presented as a pattern language, a collection of patterns related to a central concept, and given to students as a printed handout. A PDF version was also made available via the class website. Many students commented that a hypertext format would have been much easier to navigate and use when working on the class assignments. I also felt that a hypertext version would also provide the opportunity to illustrate the relationships between the elements of the $P^3\cal{F}$. A web-based presentation could also be more interactive and visually engaging. Therefore, I made the decision to design and implement a $P^3\cal{F}$ web site.

Key design requirements for the web site were a clear and logical organization, straightforward navigation throughout the site, and consistent presentation on different platforms and web browsers. I chose to use a banner-top, left-menu layout with expanding submenus as the basic layout for the web site, as shown in Figure 4.1. I experimented with various top-menu layouts to maximize the available content area, but the fifteen-item submenu for the principle patterns would flow into two or three lines unless the viewer was using a wide, high-resolution display. Multiple lines in a single submenu can be confusing, and this layout was also very sensitive to a user’s font size selection and the size and shape of the browser window. The left-menu layout minimized these problems while allowing context-dependent expansion of submenus to simplify navigation of the site.

While the basic layout was sufficient for most of the web site, I wanted the principle patterns to have something other than a “list of elements” structure. In particular, I wanted the layout of the pattern layout to reflect the underlying organization of the pattern structure. The layout I finally settled upon is shown in Figure 4.2. This layout adds two iconic elements to the pattern structure described in Section 3.3.2: a sketch (top left corner) and a graphic (top right corner). The sketch for each principle pattern is an adaptation of the icon Alexander associates with each of the fifteen properties and transformation in the Nature of Order. Each graphic is a fractal image I created to capture, in a limited way, vision of what the principle means to me. The sketch and graphic are placed on either side of the block containing the Principle Name and statement to help visually bind them to the name.

Continuing down the principle pattern, the Forces and Counterforces are placed on either side of the layout, indicating the oppositional nature of these two pattern elements. The forces and counterforces are balanced by the Intent, which represents how and why the conflict and tension can be resolved through the application of this principle. A list of Metaphors is also placed between the forces and counterforces because these analogical names also serve to illustrate the essential conflict and resolution embodied in the principle pattern. Related Principles provides the viewer with a list of links to other principle patterns in the $P^3\cal{F}$ that are closely related to the current one, and the Discussion and Examples sections of the
layout provide additional details about the principle, why it is important, and how it is used to strengthen the overall design of a system.

4.2.1.2 Practice & Design Problems

Think aloud studies are commonly used for investigating how subjects attempt to solve complex problems. Because continuously verbalizing your own thoughts is not a natural behavior for most people, it is important to provide “training” exercise(s) to help acclimate subjects to thinking out loud while working[96, 98]. The exercise should be kept relatively simple so that the subject can “concentrate” on verbalizing his or her thoughts and become comfortable with this behavior. For this study, I chose two problems to use as a training exercise, shown in Figure 4.3 with the instructional text provided to the subjects.

Subjects were introduced to the think aloud procedure during the first observation session.
They were given the practice problems and the instructions were reviewed, emphasizing that they should just try to say what they are thinking without trying to explain or interpret their thoughts. They were then asked to work through Practice Problem 1 while verbalizing their thoughts. All of the subjects had some degree of difficulty, so I worked the problem while thinking out loud to illustrate the procedure. They were then asked to work on the second problem while thinking aloud, and all of them performed significantly better. The time spent on the practice problems was not recorded.

Before searching for potential design problems to use in the study, I determined several key requirements for the problems. First, they should be realistic, non-trivial software system design scenarios that could not be effectively solved by a simple, single-user program. Second, I needed problems from a variety of application domains and business settings to help minimize the effects of a subject’s bias towards or against problems of a particular type. The scenarios should be described in business terms rather than a list of explicit requirements and specifications, and should incorporate different perspectives (e.g., users, management, and customers). The problems should also have a degree of ambiguity created by missing information, conflicting statements, and/or extraneous comments. This ambiguity should motivate the subjects to ask questions to clarify the problem and fill in missing details. Finally, all of the problems chosen for the study should have approximately the same level of difficulty and complexity.

Recognizing that the business setting of the problems could potentially cause subjects to
**Instructions:**

One of the major goals of this research study is to develop a better understanding of how computer science students approach and solve complex software design problems. A key part of this understanding will result from observing you work on solutions to real-world design problems. This observation will be made through audio-video recordings, and to help capture what you are thinking and doing, you are encouraged to “think aloud” while you work.

Thinking out loud means that you should keep talking while you work, speaking whatever thoughts come to mind no matter how unimportant they may seem to you. Do not try to explain or interpret your thoughts to the camera or the researcher, just work on the problem and say out loud what comes to mind as you work.

**Practice Problem 1:** A bottle of wine costs $12.50. The wine costs $10.25 more than the bottle. How much does the bottle cost?

**Practice Problem 2:** Design an efficient algorithm to sort an array of N integer values. The algorithm must sort the array in place without using any additional memory. The initial state of the array before sorting will be semi-sorted in reverse order at least 63% of the time.

Figure 4.3: Think aloud training exercise

be confused about a particular problem, I decided to allow them to choose the problems they would be observed working on. However, to maintain control of the experiment and to generate data that would be comparable across the subjects, the choices were limited to two problems at each session. Since there were four observations, the experiment would require a total of eight software system design problems. A variety of software engineering and system analysis and design textbooks ([36, 83, 84, 175, 201, 204, 219, 245, 253]) were reviewed to identify candidate problems. Twenty-three problems were initially selected from running examples, advanced problems, and project suggestions in the textbooks. This set was iteratively refined and pruned down to the necessary eight problems, which were then paired up to produce four problem sets which are listed in Appendix A. Each problem set was prefaced with the instructions shown in Appendix B.

To assign problem sets to the research subjects for each observation session, a set of randomly generated integer sequences was used, with each sequence containing four unique integers between 1 and 4, inclusive. The positions within the sequence were mapped to the four observation sessions and the integers mapped to problem set numbers. The assignments were made based on a planned maximum of twelve subjects. The resulting problem set to session and subject assignments are shown in Table 4.2.
4.2.1.3 Design Skills Self-Assessment Questionnaire

As noted earlier in this chapter, the purpose of the self-assessment questionnaire is to identify changes in the subjects’ perceptions of their own design skills and strategies over the course of the experiment. To quantify these perceptions, the questionnaire asked subjects to indicate their agreement or disagreement with statements describing their strategies and behaviors while working on complex software design problems. The statements were constructed based on the design behaviors and strategies of experts and novices identified in the literature review and pilot study.

The initial set of statements were evaluated by a group of doctoral students (a “dissertation support group”) that I met with on a regular basis. Critiquing each other’s work was a normal activity for the group, and evaluating surveys and similar research instruments was the type of help provided most frequently. The statements were refined through several iterations based on the feedback from my colleagues. Over the course of these refinements, the statements were grouped based on the three general behavior types described in the literature: viewing the design problem, navigating the problem/solution space, and making design decisions. The statements were also developed into phrases completing a common sentence fragment. Subjects indicated the level of their agreement or disagreement with each completed sentence using a five-point Likert scale\[167\]. The common sentence fragment for the statements about viewing the design problem was “To help me create and maintain my view(s) of a large and complex software design problem, I usually...” The phrases completing this sentence are listed below, grouped by expert or novice orientation, although they were mixed in the questionnaire given to the subjects.

Expert

- incorporate the application context, the computational domain, and views of all stakeholders.
- recognize abstract experiences and patterns that can be applied to the current problem.
- use metaphors, common names, and other abstract labels to simplify the problem and solution.
construct solution models that incorporate multiple levels of detail.
• create models that can be mentally simulated to evaluate possible solutions.

Novice
• maintain a focused view of only the part of the system I am working on, guided by my personal perspective and experience.
• develop solution models using pseudocode, programming languages, or other tools that generate program code.
• try to reorganize or restructure the problem so that it resembles problems I have solved before, allowing my to reuse those specific solutions for the new problem.
• am concerned more about getting a working solution running than about which issues are more or less important than others.

Expert
• maintain a flexible process that allows me to opportunistically choose what part of the system I am working on at any time.
• have a high tolerance for uncertainty and ambiguous information.
• am able to consider multiple issues of elements simultaneously.
• regularly and critically review my design work in progress and the way that it was developed.
• define parts of the problem or solution based on a boundary that describes relationships between those parts.

Novice
• define parts of a system in terms of their functionality within their structural location in the system.
• concentrate on addressing one issue or element of the problem at a time.
• have access to concrete examples of requirements that incorporate details and information that are as complete as possible.
• control my design process by keeping it centered on my own knowledge and experiences.
• follow a pre-defined sequence of steps to move from problem to solution, using a trial and error approach to solve each part of the problem.

The common sentence fragment for the statements about navigating the design space was “As I move from a complex problem to a designed solution, I am more productive when I...” which was completed by the following phrases:

Expert
• maintain a flexible process that allows me to opportunistically choose what part of the system I am working on at any time.
• have a high tolerance for uncertainty and ambiguous information.
• am able to consider multiple issues of elements simultaneously.
• regularly and critically review my design work in progress and the way that it was developed.
• define parts of the problem or solution based on a boundary that describes relationships between those parts.

Novice
• define parts of a system in terms of their functionality within their structural location in the system.
• concentrate on addressing one issue or element of the problem at a time.
• have access to concrete examples of requirements that incorporate details and information that are as complete as possible.
• control my design process by keeping it centered on my own knowledge and experiences.
• follow a pre-defined sequence of steps to move from problem to solution, using a trial and error approach to solve each part of the problem.

The common sentence fragment for the statements about making design decisions was “When I am working on a complex software design problem, decisions I make about the problem and how to solve it are...” and was completed by the following phrases:
Expert

- confident, reflecting my awareness of the consequences of my decisions.
- rarely final and are open to alternatives and changes as the design progresses.
- often use “soft” or subjective information as a means for building my understanding of the purposes and goals for the system.
- based on first principles or proven best practices of the problem and application domains.
- creative and often do not logically follow from the information that is available.
- oriented towards the overall purposes and goals for the system.

Novice

- solutions that follow directly from the information available at that time.
- strongly grounded in my personal experience.
- tentative and uncertain until they are written as executable code and evaluated.
- directed towards meeting the explicitly stated requirements as I understand them.
- closed to change or modification once decided.
- confidently made when I have numerical information available.

The iterative refinement and development of the survey instrument with the assistance of my dissertation support group helped to assess the content validity of the questionnaire. Since the particular perspective of this research is new, and there are no other survey instruments available to provide a comparative evaluation, content validity is the only form of preliminary validity evaluation available\[152\]. While the expectation is that the research subjects responses will change over time as a result of the experimental treatment, the reliability of the instrument was also evaluated. Eight members of the support group answered the final version of the questionnaire two times, two weeks apart. The correlation coefficients calculated from these two sets of responses ranged from 0.73 to 0.89, indicating that the test-retest reliability of the questionnaire is good\[196\]. The final version of the questionnaire, as administered in the study, is shown in Appendix C.

4.2.1.4 Exit/Feedback Questionnaire

The exit/feedback questionnaire was designed to collect data from the research subjects about their experiences while participating in the research study. Three groups of questions were developed and refined to gather feedback on the Principles, Patterns, and Process Framework itself, the $P^3F$ web site, and the design problems used in the study. Space was also provided to allow the subjects to provide additional comments or suggestions on any aspect of the study. The questionnaire was administered at the end of the final observation session. The complete version of the exit/feedback questionnaire can be found in Appendix D.
4.2.2 Study Implementation

This research study was implemented as a interrupted time-series quasi-experiment with four observations conducted at two to three week intervals. Each observation session included the subject completing a design skills self-assessment questionnaire and solving a complex, open-ended software system design problem. During the problem-solving part of the sessions, I acted as both the observer and as a representative of the client described in the design problem. The subjects were audio-video recorded while working on the problems and encouraged to verbalize their thoughts in the process. The artifacts produced by the subjects to represent and describe their proposed solutions were also collected for analysis. The remainder of this chapter discusses the recruitment of research subjects, the experimental setup employed, and the specific protocols used for each of the four observation sessions.

4.2.2.1 Research Subject Recruiting and Selection

As noted earlier in this chapter, the study was designed for eight to twelve participants. The target population for research subjects was advanced undergraduate students in the Computer Science program at North Carolina State University. Specifically, I wanted to recruit students who had completed the undergraduate Software Engineering course (CSC326) because these students would have had some exposure to large software systems as well as to software development methodologies used in professional practice. I felt that students without this exposure might be overwhelmed by the complexity of the design problems developed for the study, negatively affecting the results.

The initial recruiting effort was a three-pronged effort. Emails were sent to students who had completed CSC326 in the previous two semesters. Students in the Senior Design capstone course, for which CSC326 is a prerequisite, were also recruited via email. I also conspicuously placed flyers with my email address and the URL of a web page describing the study on bulletin boards in areas frequented by computer science undergraduates. In addition to a description of the research, potential subjects were informed of the time requirements for participating (four sessions approximately two hours each, two to three weeks apart) and that they would be compensated at the rate of $10 per hour for their time. Four undergraduate students and three graduate students responded to this initial round of recruiting. The graduate students were informed that the compensation funding was limited to undergraduate students at that time, but that I would keep their information on file in case that situation changed.

After consulting with two members of my dissertation committee, I decided to relax the CSC326 prerequisite to allow students who were either taking the course in the current semester or who had completed all of the prerequisites for CSC326 to participate in the study. Additional recruiting emails were sent to members of several student organizations in the department via
their officers. With the instructors’ permission, I also made brief recruiting presentations to the current CSC326 and Ethics in Computing (CSC379) classes. I received five responses from undergraduate students as a result of this second recruiting push. However, all four of the undergraduates who initially responded declined to participate, citing time conflicts with the senior design course work or part-time jobs.

During this time, I also received five more responses from graduate students who wanted to participate in the study. The restrictions on the compensation funding were relaxed on the condition that there would be more undergraduates participating than graduate students, allowing me to invite the graduate students who had previously expressed interest to participate. In addition to the five most recent inquiries, two of the three graduate students who expressed interest earlier still wanted to participate.

In order to compensate the participants, they had to be “hired” by the Computer Science department as temporary employees. Prior to the first observation sessions, I met with each of the potential participants to complete the necessary paperwork that included the informed consent form and departmental employment forms, which were then forwarded to the department for verification. All five of the undergraduate students were verified and hired to participate in the study. Five of the seven graduate students could not be hired due to restrictions related to their other on-campus part-time employment. One undergraduate and one graduate student withdrew from participating in the study after the first observation session. The students who completed all of the study activities included one graduate and four undergraduate students. Two of the undergraduates were taking CSC326 concurrent with their participation in the study. The other two undergraduates had not taken CSC326, but had completed all of the prerequisites for the course. One of the participants was female.

4.2.2.2 Experimental Setup

The observation sessions were held in a private office to avoid interruptions and minimize extraneous noise that would interfere with the audio recordings. A desk was set up with an Internet-connected computer workstation, paper, pencils, and pens as shown in Figure 4.4. The camera position was important for several reasons. First, it needed to be placed to clearly record a subject’s voice even when speaking quietly. Second, the camera had to be placed to capture the subject’s activities such as writing and sketching of typing on the computer. Finally, to preserve the subject’s anonymity, the camera had to be aimed so that the subject’s face would not be visible in the recordings. Figure 4.5 is an image extracted from a session recording illustrating the camera’s recording view.

The software applications initially provided on the computer workstation (as listed on the Problem Set Instructions shown in Appendix B) were based on the assumption that the
subjects would have completed CSC326 and had experience with professional-level software development environments. During the initial selection interviews and the first observation sessions several of the participants suggested additional open-source software modeling and development applications that they had experience using in their courses. In order to provide them with familiar tools, these applications were installed on the workstation between the first and second sessions. The full collection of software tools available for the subjects to use is listed below:

**Integrated development environments:** Microsoft Visual Studio$^\text{TM}$, Eclipse Helios$^\text{TM}$ with C++, Java$^\text{TM}$, Javascript$^\text{TM}$, JEE, Modeling, and PHP$^\text{TM}$ distributions

**Source code editors:** Notepad++, jEdit

**Interactive language shells:** Python$^\text{TM}$, Ruby

**Database design:** MySQL Workbench$^\text{TM}$, DBDesigner 4, Database Design Tool

**Graphical modeling software:** Microsoft Visio 2007$^\text{TM}$, ArgoUML, StarUML$^\text{TM}$, Dia

**Command-line compilers:** Java$^\text{TM}$ v 1.6, GCC v 4.5

**Office productivity software:** OpenOffice$^\text{TM}$ 3, Microsoft Office$^\text{TM}$

### 4.2.2.3 Observation Session Protocols

In order to maintain consistent execution of the study throughout all of the observations, protocols for each observation session were developed and compiled as scripts and checklists.
Each script/checklist included blank spaces for recording the date, start and end times, and subject ID. At the conclusion of each session, any hand-written artifacts were attached to the checklist for that session. Electronic documents were also printed and attached to the checklist. The four protocols are listed below.

**Session 1:**

- Record the session start time.
- The subject is introduced to Self-Assessment Questionnaire, giving him or her time to read through the instructions. Explain that the intent of the questionnaire is to develop a “picture” of how they view their own design skills and attitudes. Emphasize that there are no right/wrong or good/bad answers. Tell him/her that they can ask questions if they need to clarify any of the questionnaire statements. The subject is given time to complete the questionnaire, after which it is collected and set aside.
- Subject is given the think aloud practice worksheet and allowed to read the instructions. Any questions the subject has about the procedure are answered. Emphasize to the subject that they are to try to say what they are thinking without explaining or interpreting their thoughts. Subject works through the first problem to practice thinking out loud. Encourage as needed. If subject has problems verbalizing their thoughts, demonstrate the procedure by working through the first problem myself. Allow subject to ask questions or comment on the procedure before starting on second problem. The subject is asked to work on the second problems, thinking aloud with encouragement as needed.
- Explain the observation recording process to the subject, noting the position of the video camera and demonstrating that it is positioned to avoid capturing their face unless they lay their head down on the keyboard. Subject is also shown the resources
available for documenting their solutions (software installed on workstation, paper, pencils, and pens).

- Subject is presented with the first of the randomly selected design problem sets and given time to read through the instructions and both of the problems. Review the think aloud procedure with the subject. Explain that the problems are intended to be difficult and that complete solutions are not expected in the one-hour time available. Explain that when they are working on the problem, they may ask me questions as either the research observer (procedural questions) or as a representative of the client described in the problem. Give the subject time to ask general questions about both problems to help him or her decide which one they want to work on. Subject is asked to select the problem, indicating their selection by circling or underlining the corresponding problem number on the problem sheet.

- Start the video recorder and tell the subject to begin working on the selected problem. Record the start time.

- After one hour, tell subject to stop working and to save any electronic documents to the “My Documents” folder on their desktop. The video recording is stopped and any paper documents are collected and set aside with the Self-Assessment questionnaire. Record the stop time.

- Schedule the next observation session at least two weeks but not more than three weeks from today’s date. Record the date and time. Reassure the subject that if something unexpected comes up the next session can be rescheduled. Record the stop time for the session.

- Ask the subject to open a web browser and navigate to the “PayDay” online timesheet management application via the Computer Science department’s “employment forms” web page. Ask the subject to login and record their time for this session. Let them know when the current pay period ends and the deadline for turning in their timesheet. If time allows, ask them to print their timesheet now.

- Thank the subject for their time and participation.

- Send subject an email to confirm the next scheduled session date and time, and thank them again for their participation. (completed within one business day)

**Session 2:**

- Record the session start time.

- Briefly reiterate the research being done. Note the similarities in the activities he or she did in the first session with those that will be performed in the second and subsequent sessions.

- Review the instructions for the Self-Assessment Questionnaire. Emphasize that they should not try to remember their responses from the previous session, but should answer based on their thoughts at this time.

- Subject is presented with the second of the randomly selected design problem sets and given time to read through the instructions and both of the problems. Review the think aloud procedure with the subject and that complete solutions are not expected
in the one-hour time available. Reiterate that when they are working on the problem, they may ask me questions as either the research observer (procedural questions) or as a representative of the client described in the problem. Give the subject time to ask general questions about both problems to help him or her decide which one they want to work on. Subject is asked to select the problem, indicating their selection by circling or underlining the corresponding problem number on the problem sheet.

• Start the video recorder and tell the subject to begin working on the selected problem. Record the start time.

• After one hour, tell subject to stop working and to save any electronic documents to the “My Documents” folder on their desktop. The video recording is stopped and any paper documents are collected and set aside with the Self-Assessment questionnaire. Record the stop time.

• Introduce the subject to the Principles, Patterns, and Process Framework. Show them the web site, highlighting how it is organized. Discuss each of the principles individually, referring to the pattern-based presentation on the web site. Show them the iterative design decision-making process and talk about how it is intended to help guide their thinking as they work on design problems. Answer any questions the subject has throughout the discussion.

• Demonstrate how the framework is used by working on the problem the subject did not choose to work on in the recorded observation. Work through several iterations of the process, encouraging the subject to take a leading role in developing a solution. Encourage the subject to try to apply the framework to design-related problems they work on in their normal coursework as additional practice before the next session.

• Schedule the next observation session at least two weeks but not more than three weeks from today’s date. Record the date and time. Reassure the subject that if something unexpected comes up the next session can be rescheduled. Record the stop time for the session.

• Ask the subject to login and record their time for this session in the PayDay application. Let them know when the current pay period ends and the deadline for turning in their timesheet. If time allows, ask them to print their timesheet now.

• Thank the subject for their time and participation.

• Send subject an email to confirm the next scheduled session date and time, and thank them again for their participation. Be sure to include the URL for the web site. (completed within one business day)

Session 3:

• Record the session start time.

• Start the session with a review of the Principles, Patterns, and Process Framework. Give the subject time to ask questions or make comments about the framework, then step through two more iterations of the process using the problem from the previous session.
• Ask the subject to complete the Self-Assessment Questionnaire, reminding them that they should not try to remember their responses from the previous session, but should answer based on their thoughts at this time.

• Subject is presented with the third of the randomly selected design problem sets and given time to read through the instructions and both of the problems. Review the think aloud procedure with the subject and that complete solutions are not expected in the one-hour time available. Reiterate that when they are working on the problem, they may ask me questions as either the research observer (procedural questions) or as a representative of the client described in the problem. Give the subject time to ask general questions about both problems to help him or her decide which one they want to work on. Subject is asked to select the problem, indicating their selection by circling or underlining the corresponding problem number on the problem sheet.

• Start the video recorder and tell the subject to begin working on the selected problem. Record the start time.

• After one hour, tell subject to stop working and to save any electronic documents to the “My Documents” folder on their desktop. The video recording is stopped and any paper documents are collected and set aside with the Self-Assessment questionnaire. Record the stop time.

• Schedule the next observation session at least two weeks but not more than three weeks from today’s date. Record the date and time. Reassure the subject that if something unexpected comes up the next session can be rescheduled. Record the stop time for the session.

• Ask the subject to login and record their time for this session in the PayDay application. Let them know when the current pay period ends and the deadline for turning in their timesheet. If time allows, ask them to print their timesheet now.

• Thank the subject for their time and participation.

• Send subject an email to confirm the next scheduled session date and time, and thank them again for their participation. Be sure to include the URL for the $P^3F$ web site. (completed within one business day)

Session 4:

• Record the session start time.

• Review the $P^3F$, allowing the subject to ask questions and make comments.

• Ask the subject to complete the Self-Assessment Questionnaire, reminding them that they should not try to remember their responses from the previous session, but should answer based on their thoughts at this time.

• Subject is presented with the third of the randomly selected design problem sets and given time to read through the instructions and both of the problems. Review the think aloud procedure with the subject and that complete solutions are not expected in the one-hour time available. Reiterate that when they are working on the problem, they may ask me questions as either the research observer (procedural questions) or as a representative of the client described in the problem. Give the subject time to
ask general questions about both problems to help him or her decide which one they want to work on. Subject is asked to select the problem, indicating their selection by circling or underlining the corresponding problem number on the problem sheet.

- Start the video recorder and tell the subject to begin working on the selected problem. Record the start time.
- After one hour, tell subject to stop working and to save any electronic documents to the “My Documents” folder on their desktop. The video recording is stopped and any paper documents are collected and set aside with the Self-Assessment questionnaire. Record the stop time.
- Thank the subject for his or her participation in the study. Give them the Exit & Feedback Survey and explain that this is distinct from the data collected over the course of the study. Indicate that unlike the Self-Assessment Questionnaire, their responses to this survey are anonymous and will be maintained separately from the other study data.
- Ask the subject if they would like to request more information about the $P^3F$, noting that this personal information will not be part of the study data and will not be used in any analysis or discussion of the study.
- Ask the subject if they will consent to being contacted for a possible follow-up study to be conducted in the future (approximately twelve to eighteen months). Emphasize that they are not consenting to participate, only to be contacted and given the opportunity to be part of the future study. They will have the option of accepting or declining that participation at that time.
- Ask the subject to login and record their time for this session in the PayDay application. Let them know when the current pay period ends and the deadline for turning in their timesheet. If time allows, ask them to print their timesheet now.
- Thank the subject for their time and participation, and explain that their study-related activities are now complete.

### 4.3 Pre-Analysis Data Processing

This section describes the procedures used to prepare the raw data for analysis and evaluation. As noted earlier, three distinct types of data were collected in this research study: questionnaire responses, verbal protocols (as audio/video recordings), and artifacts produced as solution representations. The design skills self-assessment questionnaire data was suitable for analysis in its raw form, and was recorded in an electronic spreadsheet to simplify the use of analytical tools. The exit/feedback survey responses were also recorded in another spreadsheet. The recordings and solution artifacts required additional processing in order to prepare them for analysis.
4.3.1 Audio/Video Recordings

The audio/video recordings are a rich data source, but before they can be analyzed they must first be transcribed and coded. Transcription of the recorded audio to text is performed to simplify the analysis process, particularly in the coding procedure where fragments of the recordings must be selected for association with one or more codes. In addition to capturing the recorded verbalizations, notes were also inserted into the transcripts to indicate long (greater than two to three seconds) episodes of non-verbalized thinking. Actions performed by the subjects, such as writing, drawing, using the computer, and other non-verbal activities related to the problem-solving process, were also noted in the transcripts.

Two additional processing steps are commonly recommended before coding verbal protocols: cutting the protocols into smaller pieces and distributing these to the coders randomly\[248\]. To accomplish this, the recordings were split into approximately fifteen-minute sections, using non-verbal passages of at least five seconds as cut points. The transcript text files were also split at these points, copying the non-verbal notation at the end of the leading section to the beginning of the trailing section. To further obfuscate the individual subjects and the ordering of the protocol sections, each section recording and transcript file was renamed with a randomly-generated alphanumeric filename.

The final pre-analysis processing was the coding of the protocols. The remainder of this section discusses the coding process, starting with the development of the coding scheme. This is followed by a discussion of the procedures used to implement the protocol coding.

4.3.1.1 Design and Development of the Coding Scheme

In verbal protocol analysis, the coding scheme is a mapping between the recording of what a subject says and does while performing some task and the cognitive processes of interest to the researcher. In this study, the cognitive processes of interest are the design decision-making processes of the research subjects, specifically how subjects make their decisions and why they make the choices they do. Based on this, the coding scheme was designed to identify and describe design decisions in the recorded protocols. The codes that evolved during the pilot study analysis formed the foundation of the coding scheme used in this study.

The pilot study coding scheme characterized design decisions based on the action performed to solve the identified problem. The system entity under consideration was also explicitly recorded in each decision coding, as were any alternatives and the reason the subject chose the particular solution. For this study, a coding scheme using four variables was designed to capture the same kind of information about design decisions, but to do so with greater granularity and specificity. The four variables used in this scheme are:
Level - The level of abstraction or detail within the design of the system where the decision is made.

Type - The type of design decision made.

Rationale - The influences and/or forces that were the basis for the decision and the choice made.

Subject - The subject or object of the decision and any alternative solutions identified.

Individual design decisions in isolation are not clearly indicative of novice or expert behavior when taken in isolation, but when viewed as sequences of decisions, larger-scale patterns of behavior can become evident. For example, a series of decisions that may proceed in a linear fashion from a conceptual statement to a particular concrete entity without considering the side effects or unknown influences adjacent to that path. Another example might be a cyclic alternation between defining a problem, proposing a solution, testing that proposal, and evaluating the results to feed back into refining the problem definition and/or solution proposal.

The codes associated with the Level variable describe a range of abstraction or detail level from abstract conceptual ideation to specific, concrete details of the proposed design. Observing the changes in the levels in a sequence of design decisions can provide insights into how subjects build, manipulate, and navigate the design and problem space as they work towards a solution. Kokotovich noted that novices have a tendency to rush to concrete implementation[154], while expert designers make greater use of abstractions and mental models to understand the problem and potential solutions[6, 51, 90]. The decision level codes and their definitions are listed in Table 4.3.

The second coding variable identifies the Type of a design decision. Code values for this variable distinguish the key action or activity performed by the subject, either in the process of making the decision or as a result of it. The Level and Type codes together provide the means to track the location and direction of a subject’s thinking as they design and refine a solution. Table 4.4 lists the code values for the Type variable.

For this study it is also important to try to understand why a particular decision was made. To capture this information in the coding process, a Rationale variable is included in the coding scheme. Code values for the Rationale variable will characterize the justification or basis for making a decision and are listed in Table 4.5.

The final coding variable, Subject, is used to record information about the subject or object of a design decision. Combined with the Level and Type variables, the Subject provides the capability of locating a design decision in the problem/solution space. This will allow patterns of movement through the design space to be identified and studied, illuminating larger-scale behavioral patterns which may then be categorized as expert-like or novice-like.

131
Table 4.3: Decision Level Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual (L-Con)</td>
<td>Understanding and characterizing the problem, focusing on concepts and ideas that help build understanding.</td>
</tr>
<tr>
<td>Requirements (L-Req)</td>
<td>Defining, analyzing, and refining stated requirements and eliciting additional requirements.</td>
</tr>
<tr>
<td>System (L-Sys)</td>
<td>Describing and refining global characteristics of the system when viewed as a whole, complete unit.</td>
</tr>
<tr>
<td>Component (L-Com)</td>
<td>Distinguishing and developing major component parts of the system.</td>
</tr>
<tr>
<td>Detail (L-Det)</td>
<td>Defining and evaluating low-level, concrete details necessary to implement the proposed solution.</td>
</tr>
<tr>
<td>Interaction (L-Int)</td>
<td>Defining, describing, and analyzing interactions between the system and various agents that use or are used by the system. These agents include human users as well as software-based entities that interact with the system.</td>
</tr>
<tr>
<td>Environment (L-Env)</td>
<td>Understanding and characterizing the systems execution environment(s).</td>
</tr>
</tbody>
</table>

Other utterances and actions not associated with solving the design problem may occur, and these also need to be coded. For example, a subject might choose to stop working on the problem, make unrelated comments, or be thinking without verbalizing their thoughts. To identify these situations, five additional codes were added to the overall coding scheme. These codes are listed in Table 4.6.

4.3.1.2 Coding Procedures

Two individuals with extensive experience coding verbal and textual protocols were hired to perform the coding for this study. To help ensure objective and reproducible coding, neither coder had any connection or interest in the outcome of this research[248]. The coders were given the coding scheme and instructions, shown in Appendix E. We then met as a group to discuss the scheme and to answer any questions about my expectations.

The second part of this meeting was devoted to training and consensus-building on the coding process. Both coders performed an initial and independent coding of the same verbal protocol piece. We then reviewed both codings, discussing the differences and similarities between their codings and how they fit with my general expectations. A second piece was then coded and reviewed. At this point, both coders felt they had a solid and common understanding of the coding scheme and how to apply it to the protocols.

The Atlas.ti™ qualitative analysis software was used to organize the protocol pieces and
Table 4.4: Decision Type Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition (T-Dec)</td>
<td>Separate a problem into smaller parts with simpler solutions that can then be recombined to solve the larger problem.</td>
</tr>
<tr>
<td>Differentiation (T-Dif)</td>
<td>Distinguish between different parts or elements of a problem or system, usually through the definition of boundaries or other inter-element relationships.</td>
</tr>
<tr>
<td>Abstraction (T-Abs)</td>
<td>Generalization of problems or design elements, used to classify, categorize, or simplify an aspect of the problem or design.</td>
</tr>
<tr>
<td>Instantiation (T-Ins)</td>
<td>Assignment of specific details and/or unique attributes to make a particular design element more concrete or specific.</td>
</tr>
<tr>
<td>Composition (T-Com)</td>
<td>Combining solutions to smaller problems to form a solution to a larger problem; the inverse of decomposition.</td>
</tr>
<tr>
<td>Integration (T-Int)</td>
<td>Similar to composition, but using a larger-scale perspective structuring a solution (or set of solutions) within the context of the system as a whole (or of a larger part of the system).</td>
</tr>
<tr>
<td>Postponement (T-Pos)</td>
<td>The need for a particular design decision has been identified but deferred in favor of another (usually easier or less risky) decision or because there is not enough information to make an informed decision at this time.</td>
</tr>
<tr>
<td>Proposal (T-Pro)</td>
<td>Suggesting or hypothesizing a particular solution or solution fragment.</td>
</tr>
<tr>
<td>Information Gathering (T-IG)</td>
<td>Seeking out additional information not included in the problem statement in support of the design process.</td>
</tr>
<tr>
<td>Evaluation (T-Eva)</td>
<td>Testing a solution or solution fragment in the context of the problem it is proposed to solve.</td>
</tr>
</tbody>
</table>

transcript text files and to record the codings. This software allows codes to be applied to either the audio/video recording or to the text transcript. Both coders preferred to apply the coding to the text files as these were easier to mark the start and stop points of the coded text.

Protocol pieces (video recording and transcript) were distributed to the coders in lots of sixteen at one to two week intervals. Pieces were assigned to these lots randomly, such that the two coders did not have the same data to code at any time in the process. After the first lot was completed, I reviewed the codings to check that they were consistent with the first two pieces and were sufficient for the intended analysis. As both coders completed coding a particular piece, the codings were merged in order to prepare them for further analysis. When all of the pieces of a particular observation were completed, they were merged to reform the complete recording and transcript for that session.

To assess the consistency and reliability of the coding process, Fleiss’ kappa was calculated using the Coding Analysis Toolkit (CAT), an online (http://cat.ucsur.pitt.edu/default.

133
Table 4.5: Decision Rationale Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements or Problem Statement (R-RPS)</td>
<td>Information explicitly stated in or logically deducible from the given problem statement and/or requirements is used as the basis for a decision.</td>
</tr>
<tr>
<td>Application Knowledge (R-AK)</td>
<td>Recall and use of previously gained knowledge about the application or the environment in which the designed artifact is to be used, e.g., database, web server, user interface, etc.</td>
</tr>
<tr>
<td>Domain Knowledge (R-DK)</td>
<td>Recall and use of knowledge about the problem domain, e.g., business processes or rules, accounting procedures, regulations, etc.</td>
</tr>
<tr>
<td>Best Practices or Design Principles (R-BP)</td>
<td>Specific, documented principles or practices applicable to the problem and context justify a particular choice.</td>
</tr>
<tr>
<td>Formal or Experimental Analysis (R-FEA)</td>
<td>The decision is a result of a logical analysis of alternatives or a mathematical proof.</td>
</tr>
<tr>
<td>External Reference Material (R-ER)</td>
<td>Look-up and use of material outside the given problem statement and the subject’s own direct knowledge and experience.</td>
</tr>
<tr>
<td>Personal experience (R-PE)</td>
<td>Decisions based on generic, non-specific personal experience, guessing, etc.</td>
</tr>
<tr>
<td>Unknown or Indeterminable (R-Unk)</td>
<td>No clear indication of the justification for a decision.</td>
</tr>
</tbody>
</table>

aspx) suite of tools designed to facilitate efficient and effective analysis of text datasets coded using Atlas.ti. Fleiss’ kappa is used to assess the inter-rater reliability between a fixed number of raters who are classifying items. This measure specifically assumes a fixed number of raters and that different items are rated by different individuals, as is the case in verbal protocol coding[101]. The overall \( \kappa \) statistic for all codes and protocols was computed to be 0.52. Table 4.7 lists the \( \kappa \) for each of the codes used.

It has been noted that “no one value of kappa can be regarded as universally acceptable” due to differences in coding schemes, coders or observers, and the quality of the coded text[23]. Landis and Koch[162] suggest that \( \kappa \) values of 0.41 to 0.60 indicate moderate agreement between coders or observers, while Fleiss characterizes values of 0.40 to 0.75 as fair to good[102, p. 218]. As shown in Table 4.7, \( \kappa \) values for individual codes ranged from 0.41 to 0.89, which are acceptable for this study. Exact 100% matching was not expected in this coding for several reasons. First, the Atlas.ti software used in the coding process requires coders to define quotations to which the codes are applied. A difference as small as one character between two quotations is recorded as a non-exact match or overlap which is weighted differently than an exact match in the CAT computation. Also, several of the codes had similar definitions (e.g., Decomposition and Differentiation, Integration and Composition), and while the distinction is generally obvious to individuals in the software engineering discipline, it may be less so to those whose expertise
lies in other areas, such as the coders employed for this research. There were many instances where the one coder used one term and the other applied the similar code. However, the coding did prove effective for the analysis, described in Section 5.1.2, as the differences in coding often helped to illuminate different aspects of the subjects thought processes.

4.3.2 Design Artifacts

Before considering how to evaluate the design artifacts produced by the research subjects, there must be an understanding of what constitutes a “design artifact” in this study. The intent of this research is to study how subject approach solving difficult problems, not how they try to learn how to use particular software design modeling tools and techniques. Therefore, it is important to provide the subjects with tools that allow them to design solutions in ways that they are comfortable with and that suit their abilities and existing tool knowledge. Subjects will have access to a variety of software tools such as integrated development environments (IDEs), graphical database design programs, Unified Modeling Language (UML) diagraming tools, and text editors, as well as tablets for hand-written notes, sketches, and diagrams.

Limiting subjects to the use of specific tools would introduce the risk of bias in the artifacts resulting from their experience (or lack thereof) with those tools. On the other hand, allowing subjects to represent their solutions in whatever way they prefer could complicate the evaluation process due to the variety of possible forms of representation. This could also introduce an evaluation bias as a well-constructed electronic diagram might be perceived as more professional and “correct” than hand-drawn sketches or diagrams even though the paper-and-pencil solution should be considered a better solution. This risk can be minimized through the use of clear and specific design quality criteria to measure artifact quality.

A second consideration for choosing an evaluation method is the number of different problems

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unusable Fragment (N-Un)</td>
<td>Any coded fragment irrelevant to the protocol.</td>
</tr>
<tr>
<td>Confusion (N-Conf)</td>
<td>A fragment that explicitly states that the subject is stuck or confused and does not know how to proceed.</td>
</tr>
<tr>
<td>Non-verbalized thinking (N-NV)</td>
<td>A fragment of the recorded protocol where the subject is engaged in working on the problem but is not verbalizing her/his thoughts.</td>
</tr>
<tr>
<td>Stop (N-Stop)</td>
<td>Subject has decided to stop working on an incomplete solution.</td>
</tr>
<tr>
<td>Complete (N-Comp)</td>
<td>Subject has decided that the solution is as complete as he/she can make it and will stop working on it.</td>
</tr>
</tbody>
</table>
Table 4.7: Inter-rater reliability of verbal protocol coding

<table>
<thead>
<tr>
<th>Code</th>
<th>κ</th>
<th>Code</th>
<th>κ</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-Con</td>
<td>0.55</td>
<td>R-ER</td>
<td>0.58</td>
</tr>
<tr>
<td>L-Det</td>
<td>0.58</td>
<td>R-PE</td>
<td>0.56</td>
</tr>
<tr>
<td>L-Env</td>
<td>0.46</td>
<td>R-RPS</td>
<td>0.89</td>
</tr>
<tr>
<td>L-Int</td>
<td>0.41</td>
<td>T-Abs</td>
<td>0.43</td>
</tr>
<tr>
<td>L-Req</td>
<td>0.47</td>
<td>T-Com</td>
<td>0.47</td>
</tr>
<tr>
<td>L-Sys</td>
<td>0.47</td>
<td>T-Dec</td>
<td>0.52</td>
</tr>
<tr>
<td>N-Comp</td>
<td>0.46</td>
<td>T-Dif</td>
<td>0.43</td>
</tr>
<tr>
<td>N-Conf</td>
<td>0.49</td>
<td>T-Eva</td>
<td>0.50</td>
</tr>
<tr>
<td>N-NV</td>
<td>0.48</td>
<td>T-IG</td>
<td>0.65</td>
</tr>
<tr>
<td>N-Stop</td>
<td>0.50</td>
<td>T-Ins</td>
<td>0.44</td>
</tr>
<tr>
<td>N-Un</td>
<td>0.50</td>
<td>T-Int</td>
<td>0.48</td>
</tr>
<tr>
<td>R-AK</td>
<td>0.49</td>
<td>T-Pos</td>
<td>0.67</td>
</tr>
<tr>
<td>R-BP</td>
<td>0.52</td>
<td>T-Pro</td>
<td>0.54</td>
</tr>
<tr>
<td>R-DK</td>
<td>0.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td><strong>0.52</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

subjects will work on during the observation sessions. If subjects are given the same problem in more than one session, their prior knowledge of the problem would influence the solution produced in the later session. Therefore, there will need to be enough problems available to cover the intended number of observations sessions for the duration of the study. The variety of problems will result in a greater variety of solutions for the duration of the study. The variety of problems will result in a greater variety of solutions for the duration of the study. The variety of problems will result in a greater variety of solutions for the duration of the study. The variety of problems will result in a greater variety of solutions for the duration of the study. The variety of problems will result in a greater variety of solutions for the duration of the study. The variety of problems will result in a greater variety of solutions for the duration of the study.

The third important aspect of the evaluation process that must be taken into account is who will evaluate the design artifacts. On the surface, it might seem like the researcher would be the ideal evaluator because of the familiarity with the design problems and with the overall goals of the research. However, the researcher’s personal interest in the outcome of the study increases the risk of biased evaluations as a result of this familiarity. This risk is amplified if the researcher is also the observer due to the potential for considering memories of a subject’s behavior in conjunction with the artifacts in the evaluation process. Independent evaluators who do not have a vested interest in the outcomes of the research help to avoid these risks, although bias towards or against particular problems remains.

In order to allow objective comparisons between the design artifacts before and after the introduction of the $P^3F$, a rubric identifying specific evaluation criteria is essential. To further reduce the threats to internal validity, the evaluations should be performed by individual(s) who
are not associated with the research. The degree of agreement of evaluations performed by two or more individuals can be statistically scored to determine the reliability of their evaluations.

The design artifacts were the raw data source produced by the research subjects also required additional processing to allow comparative analysis. The research question driving this analysis is that of determining if the introduction of the $P^3F$ caused any improvement in the quality of the design solutions produced by the subjects. In order to make this comparison, the quality of the artifacts must first be objectively measured. To ensure objectivity and consistency in the artifact evaluation process, two software professionals were recruited to perform the evaluations. Both individuals had over twenty years of professional experience in the design, implementation, deployment, and maintenance of large-scale software systems.

4.3.2.1 Design Quality Rubric

Objective evaluation of design solutions requires the development of objective criteria for determining the quality of a solution. An exhaustive search of the literature did not identify a compact, general-purpose set of criteria for measuring the quality of a software system design. Software quality rubrics do exist, but they are specific to a particular perspective or aspect of a system, such as its performance, security, reliability, or usability. Expanding the search to other engineering and design disciplines yielded the Design Quality Rubric used to measure design quality in a study of students in a mechanical engineering capstone design course[233]. This rubric measured design quality in four categories: requirements, feasibility, creativity, and simplicity in addition to an “overall impression” question to capture the evaluator’s overall assessment of the design.

Using this rubric as a starting point, two additional design quality categories were added to tailor the criteria more specifically to software systems: soundness and scalability. The resulting rubric was reviewed by the external evaluators, both of whom suggested the addition of a “security” category. The final software design quality rubric used to evaluate the design artifacts contained eight metrics and was defined as follows:

**Requirements** - The design meets the technical criteria and customer requirements.

This metric is intended to measure how well the proposed solution identifies and responds to the requirements described in the problem statement. Has the subject ignored or misinterpreted features or characteristics the customer explicitly stated? Have any requirements not explicitly stated in the problem but which are directly deducible from the text been identified and addressed? Did the subject incorporate any design elements that clearly violate any stated or deducible requirements?

**Feasibility** - The design is feasible in its application and implementation.
This metric is intended to measure the extent that the proposed solution is directed towards a working deliverable, e.g., whether the proposed solution is “headed in the right direction” or not. Because the subjects only have one hour to work on the problem, they are not expected to produce solutions that are detailed and complete.

**Soundness** - The design is sound and reasonable in its use of programming languages, algorithms, data structures, and existing technology.

This metric is intended to measure how reasonable the solution is in its approach to meeting the customer’s requirements. Is the proposed solution a reasonable start at a workable and reliable solution? Does the design incorporate elements that are appropriate for the given problem? Are there elements in the proposed solution that would tend to lead towards an unreliable, unmaintainable, or otherwise weak end product?

**Creativity** - The design incorporates original and novel ideas, non-intuitive approaches, or innovative solutions.

This metric will gauge the level of creativity exemplified by the proposed solution. While the given design problems are relatively common situations, they all have elements or aspects intended to set them apart from “stock,” off-the-shelf solutions. Does the proposed solution represent an attempt to find and elaborate a creative or novel solution to the problem, or has the subject tried to make the given problem fit a known solution?

**Simplicity** - The design is simple, practical, usable, reliable, maintainable, and safe, and avoiding any approaches that would introduce unnecessary complexity into the system implementation.

This metric is intended to capture the simplicity and practicality of the proposed solution. Does the solution represent the simplest solution to the problem? Has the subject added features and/or capabilities that the customer did not request, increasing the complexity of the solution? Are there unnecessary redundancies within the solution?

**Scalability** - The design is sufficient to efficiently meet reasonably expectable increases in demand and/or data volume without modification.

This metric is intended to measure how well the subject has taken into account the reasonable growth of the customer’s business as it relates to the design of their solution. Will the proposed solution be able to handle predictably significant increases in the demand for the services and information the system handles without crashing or experiencing unacceptable delays and latency? Does the proposed solution incorporate elements that would lead you to believe that the designer recognized the potential for growth in the business that would require a more robust design?
Security - The design incorporates elements sufficient to protect against security threats that would commonly be associated with the type of application and business described in the problem statement.

While the subjects participating in this study may not have had extensive and in-depth exposure to security-related topics, they should recognize the foundational need for security in business information systems. This measure should indicate, based on your evaluation of the proposed solution, the extent to which the subject has identified and addressed security-related issues in the development of their proposed solution.

Overall - Overall impression of the design solution.

This metric allows the evaluator to rate the proposed solution as a whole rather than specific aspects or perspectives on the solution. This metric is included as a separate item as a different view of the solution, one that balances and tempers the other views rather than the sum or product of the other metrics.

This rubric provides a clearly defined set of evaluation metrics that offer eight different quality measurements to be made on a solution artifact. Together, they will provide a well-rounded view of proposed solutions that will allow them to be objectively compared.

4.3.2.2 Design Artifact Evaluation Process

Subjects used both paper and electronic documents to record their solutions to the design problems. Paper documents were scanned into electronic form and combined with electronic documents when a subject used both media. Several methods of delivering the artifacts were discussed with the evaluators, including in-person meetings, postal mail, and email. We eventually decided that a web-based method would provide the greatest flexibility for the evaluators while maintaining the integrity and security of the artifacts and the evaluation data.

A PHP/SQL system was developed to deliver the design artifacts and to collect the evaluation data. The system provided a secure login to restrict access to the evaluators to protect the integrity of the collected data. A “timeout” feature was also incorporated into the system that would automatically log an evaluator out after two hours of inactivity. The login page is shown in Figure 4.6.

After successfully logging in to the system, an evaluator was presented with a list of artifacts waiting to be evaluated, as shown in Figure 4.7. Links to the design problems were also provided so that the evaluators could familiarize themselves with the problems prior to beginning their evaluations. A link to an overview of the evaluation process and a detailed description of the design quality rubric was also available from this page. To begin evaluating an artifact, the
evaluator selects the radio button adjacent to the artifact then clicks on the “Evaluate Selection” button.

After selecting an artifact, the evaluation form shown in Figure 4.8 is displayed. The artifact to be evaluated is also opened in a second browser window and the problem statement opened in a third. The problem statement is included so that the artifact can be evaluated based on the information available to the subject. The artifacts were evaluated using a seven point scale for each metric with three anchors given (1 - poor; 4 - acceptable; 7 - outstanding). A brief rationale was requested for each evaluation response for the purpose of inter-evaluator comparisons and consistency assessment. The evaluation is recorded in the database when the “Submit Evaluation” button is clicked. Alternatively, the evaluation of this artifact could be canceled without recording any of the data entered. In both cases, the child windows displaying the artifact and problem statement are automatically closed before returning the user to the evaluation home page.

After all of the evaluations were completed and the data recorded, the artifact was removed from the list of unevaluated artifacts. Evaluators were given the ability to review and modify these previously completed evaluations by selecting them from a separate list on the evaluation home page, as shown in Figure 4.9. When selected, the evaluation form, artifact, and problem statement were displayed as in the original evaluation session, but with the form populated with the most recently entered evaluation data. As before, any changes could be recorded or the re-evaluation canceled at the option of the user.

After all of the evaluations were completed, the evaluation data was exported from the database to a spreadsheet to facilitate further analysis. As with the verbal protocols, the first step was to assess the inter-rater reliability of the evaluations. Cohen’s kappa is used to
measure the agreement between two raters who classify the same set of items into a fixed number of mutually exclusive categories\[60\]. The overall $\kappa$ statistic for all metrics and artifacts was computed to be 0.8944. Table 4.8 lists the $\kappa$ for each of the metrics in the quality rubric.

Landis and Koch\[162\] note that $\kappa$ values of 0.81 to 1.0 indicate almost perfect agreement between raters, and Fleiss characterizes values over 0.75 as excellent\[102, p. 218\]. As shown in Table 4.8, the two evaluators had the strongest agreement on metrics 1, 2, and 8, and the weakest level of agreement on metrics 5 and 6. Requirements satisfaction and the feasibility of the proposed solution were expected to be the easiest metrics to evaluate, and the agreement between the evaluators supports this expectation. Simplicity and scalability, on the other hand, can be subjective decisions, explaining the lower level of agreement on these two metrics. Overall, these $\kappa$ values suggest close agreement between the two evaluators which was reinforced by their similar comments associated with their artifact evaluations.
### P3F Design Artifact Evaluation

You are currently evaluating solution "LPMBmMYT" to the "ABC, Inc. Payroll System" problem.

<table>
<thead>
<tr>
<th>Metric Number</th>
<th>Name</th>
<th>Description</th>
<th>Evaluation</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirements</td>
<td>The design meets the technical criteria and customer requirements.</td>
<td>Select...</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Feasibility</td>
<td>The design is feasible in its application and implementation.</td>
<td>Select...</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Soundness</td>
<td>The design is sound and reasonable in its use of programming languages,</td>
<td>Select...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>algorithms, data structures, and existing technology.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Creativity</td>
<td>The design incorporates original and novel ideas, non-intuitive approaches,</td>
<td>Select...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>or innovative solutions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Simplicity</td>
<td>The design is simple, practical, usable, reliable, maintainable, and safe, and avoiding any approaches that would introduce unnecessary complexity into the system implementation.</td>
<td>Select...</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Scalability</td>
<td>The design is sufficient to efficiently meet reasonably expectable increases in demand and/or data volume without modification.</td>
<td>Select...</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Security</td>
<td>The design incorporates elements sufficient to protect against security threats on would commonly associate with the type of application and business.</td>
<td>Select...</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Overall</td>
<td>Overall impression of the design solution.</td>
<td>Select...</td>
<td></td>
</tr>
</tbody>
</table>

Submit Evaluation  Cancel Evaluation

Figure 4.8: Design artifact evaluation form

Table 4.8: Inter-rater reliability for design artifact evaluations

<table>
<thead>
<tr>
<th>Metric</th>
<th>$\kappa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9396</td>
</tr>
<tr>
<td>2</td>
<td>0.9392</td>
</tr>
<tr>
<td>3</td>
<td>0.9320</td>
</tr>
<tr>
<td>4</td>
<td>0.8230</td>
</tr>
<tr>
<td>5</td>
<td>0.8137</td>
</tr>
<tr>
<td>6</td>
<td>0.8101</td>
</tr>
<tr>
<td>7</td>
<td>0.9296</td>
</tr>
<tr>
<td>8</td>
<td>0.9390</td>
</tr>
<tr>
<td>Overall</td>
<td>0.8944</td>
</tr>
</tbody>
</table>
Welcome, David

This page is your starting point for evaluating the design solution artifacts developed by the subjects in this research study. Below you will find a list of solutions that you have not yet evaluated as well as a list of those that you have. You may revise your evaluations as often as you like. At the bottom of this page you will also find links to the problem statements given to the subjects. When you select a particular solution to evaluate, you will be presented with the evaluation form and the problem statement and solution will open in separate windows so that you will have ready access to all three components while you examine and record your evaluation. You can also click on the "Evaluation Overview" below to view a detailed description of the Design Quality Rubric metrics used in the evaluation.

**Evaluation Overview**

**Unevaluated Solutions:**
- fzsI7NbP (Problem # 1)
- 456DVeRz (Problem # 1)
- 1JrKYXsQ (Problem # 5)
- IgNeLq8o (Problem # 4)
- DyQyqfTF (Problem # 7)
- dB365Opv (Problem # 1)
- WT5Juvis (Problem # 8)
- AMrC53Oi (Problem # 1)
- 5kXVIkMm (Problem # 6)
- LSWuSyIy (Problem # 3)
- rBrgtMw6 (Problem # 5)

**Previously Evaluated Solutions:**
- LPMBmMYT (Problem # 1)

**Design Problem Descriptions**
1. ABC, Inc. Payroll System
2. Birdie Masters Golf Schools Operations System
3. ACME Consulting Knowledge Management System
4. Jones Legal Investigation Service Case Management
5. The Clothing Shack Customer Management System
6. Evergreen Nursery Information Systems
7. Hoosier Burger Information Systems
8. Waiters on Call Meal Delivery Service

Figure 4.9: Design artifact evaluation home page after completing an evaluation
Chapter 5

Analysis and Results

This research project employed an interrupted time-series quasi-experiment to study the effects of introducing five novice software designers to a framework intended to help them develop expert-like design strategies. Three different sets of data were collected: 1) the subjects’ assessment of their own design skills and strategies; 2) coded verbal protocols capturing the subjects’ design decision-making processes; and 3) measures of the quality of the solutions designed by the subjects. This data and methodological triangulation is an important aspect of this research, as it enhances the rigor of the work, reduces internal and external threats to validity, and helps reduce the risk of inappropriate certainty[211].

Before presenting the results of this study, the techniques employed in the analysis of the research data are discussed in Section 5.1. Each of the five research subjects are analyzed in Section 5.2, followed by a summary discussion of these results in Section 5.3.

5.1 Analysis Techniques

The three types of data collected in this study require different analysis techniques. Overall, however, the results of this study will not be generalizable due to the number of research subjects that participated. Rather than attempting to perform detailed statistical analysis, the data are examined subject-by-subject to explore the details of the data in a qualitative manner. The overarching structure of this analysis will be to examine the self-assessments, verbal protocols, and design artifacts individually for each subject. The results of these three analyses will then be compared and contrasted to summarize what has been learned from that subject. This section describes the three types of analysis performed.
5.1.1 Self-Assessment Questionnaire Analysis

As described in Section 4.2.1.3, the self-assessment questionnaire was designed to measure how the research subject perceive their own design skills and strategies with the goal of identifying changes in those perceptions that might be associated with their exposure to the $P^3F$. The questionnaire contained statements describing strategies and behaviors exhibited by novices and experts in previous studies of designers. The statements were grouped into three categories: viewing the design problem, navigating the design space, and making design decisions. Subjects indicated their level of agreement with each statement using the scale 1) Strongly Disagree, 2) Disagree, 3) Not Sure, 4) Agree, 5) Strongly Agree. The statements are not oppositional, so that strong disagreement with a statement describing novice behavior does not imply an expert-like tendency.

Two methods were used to analyze the subjects’ responses to the self-assessment questionnaire. To study general tendencies, aggregate measures of expert and novice oriented perceptions were computed by summing the responses to the respective sets of questions in each category at each observation session to produce an assessment score for each observation-category. Abrupt changes in this score occurring after the second observation session could suggest that the introduction of the $P^3F$ may have influenced that change. Responses to individual statements were also examined to identify sharp changes over the course of the observations.

By itself, the results of these analyses are not conclusive. However, they provide insights into what the subjects think about how they approach design tasks. Also, these results, particularly those statements that the subjects strongly agree or disagree with, suggested specific behaviors to look for in the verbal protocols.

5.1.2 Verbal Protocol Analysis

The goal of the verbal protocol analysis in this study was to identify behavioral characteristics of the research subjects as they attempted to solve difficult software system design problems. As discussed in Section 4.3.1, the video recordings of the observations were transcribed and coded to produce the data used in this qualitative analysis. An iterative approach was taken to identify and classify the subjects’ observed behaviors and the order in which they occurred. These behaviors where then associated with the expert and novice strategies and behaviors identified in Chapter 2. The analysis process consisted of five phases.

Phase 1 The objective of this phase was to delineate design decisions and reconcile the subject labels for each decision if there was a difference in the coding. Two iterations over the coded transcripts were performed to be sure each decision was identified. The first pass was made in the order of the observation sessions across all of the subjects. For the second
iteration, all of the coded transcripts for a single subject examined in reverse session order. Changing the order helped to ensure a consistent inspection across all subjects and observations.

Phase 2 With the design decisions identified, delimited, and labeled with the subject of the decision, the next step was to distinguish sequences of related decisions. Two adjacent decisions were said to be related if they had the same subject/object or if the subject’s statements directly connected the decisions. Multiple reviews of each coded transcript were made to identify and confirm related decisions and connect the contiguous decision sequences. Each sequence was labeled with notes describing the subject’s larger-scale objective(s) and actions represented in the sequence.

Phase 3 In this phase, relationships between non-contiguous decisions and decision sequences were identified. The same relational criteria used in the previous phase was applied, with the requirement that the related decisions be separated by at least one unrelated decision. The objective of this phase was to capture the subjects’ wider movements through the design space. As in the previous phase, these non-contiguous decision sequences were labeled to identify the objectives and actions of the sequences. Each coded transcript was reviewed two or more times.

Phase 4 The contiguous and non-contiguous decision sequences identified in the previous two phases resulted in a collection of notes describing multiple decisions. These notes were reviewed to distill a compact set of second-level codes to label the decision sequences. These codes were then used to label the identified decision sequences based on the notes made in the previous two phases.

Phase 5 The second level coding of the transcripts was analyzed to identify patterns of behavior across two or more decision sequences. Using the strategies and behaviors identified in the literature review as a guide, these coded behavior patterns were labeled as expert-like, novice-like, or neither. The corresponding strategy or behavior was also noted.

The results of this analysis for each coded transcript was a three-level coding that highlighted and described the the expert-like and novice-like behaviors exhibited by the research subject as he or she attempted to solve a design problem. These were then compared across the four observations for each subject to identify any behavioral changes that occurred in the course of the study. Subjects’ behaviors were also compared with their self-assessment responses and with the evaluations of their solution artifacts to produce an overall view of their performance.
Table 5.1: Second-level code scheme for protocol analysis

<table>
<thead>
<tr>
<th>Group</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Context</td>
<td>Definition or refinement of the environment where the system or system element will exist. Includes non-software entities such as human activities and interactions with the system, stakeholders, hardware configuration, and business constraints.</td>
</tr>
<tr>
<td></td>
<td>Problem</td>
<td>Definition or refinement of the design problem or a particular aspect of it.</td>
</tr>
<tr>
<td>Explore</td>
<td>Depth</td>
<td>Defining and/or refining details of one aspect or element of context, problem, or solution across three or more decisions.</td>
</tr>
<tr>
<td></td>
<td>Breadth</td>
<td>Defining and/or refining aspects or parallel elements at similar levels of detail across three or more decisions.</td>
</tr>
<tr>
<td></td>
<td>Iteration</td>
<td>Revisit previous decisions or elements of context, problem, or solution with the intention of refining the results of a decision sequence.</td>
</tr>
<tr>
<td></td>
<td>Jump</td>
<td>Divergent transition from one part or aspect of context, problem, or solution to another part that is not closely related.</td>
</tr>
<tr>
<td>Develop</td>
<td>Modeling</td>
<td>Construction or refinement of abstractions or mental models of a possible solution.</td>
</tr>
<tr>
<td></td>
<td>Framing</td>
<td>Defining or refining boundaries of interfaces between entities or aspects of context, problem, or solution.</td>
</tr>
<tr>
<td></td>
<td>Concretization</td>
<td>Defining or refining specific, concrete details in a proposed solution.</td>
</tr>
<tr>
<td></td>
<td>Relationships</td>
<td>Defining or refining relationships between aspects or elements of the problem and/or solution.</td>
</tr>
<tr>
<td>Process</td>
<td>Planning</td>
<td>Defining or refining the process or approach to be used to solve the problem.</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
<td>Testing a solution or solution element in its context.</td>
</tr>
</tbody>
</table>

5.1.3 Design Artifact Quality Analysis

The solution artifacts produced by the subjects were evaluated as described in Section 4.3.2. Two sets of data were produced by this evaluation: a numeric quality score for each of the eight metrics in the Design Quality Rubric, and the comments from both evaluators explaining their rationale for the score that was assigned for each metric. Analysis of these evaluations involved comparing the evaluation scores over the four observations. The evaluations were also associated with each subject’s self-assessments and the behavioral tendencies identified in the verbal protocols.

There is also a possibility that some of the design problems may have been more difficult than others. In order to compare each subjects evaluations, the arithmetic mean ($\bar{x}$) and standard deviation ($\sigma$) were calculated for each design problem, as shown in Table 5.2.
Table 5.2: Mean evaluations for design problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Metric 1</th>
<th>Metric 2</th>
<th>Metric 3</th>
<th>Metric 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>1</td>
<td>4.80</td>
<td>1.72</td>
<td>4.50</td>
<td>2.10</td>
</tr>
<tr>
<td>3</td>
<td>3.50</td>
<td>1.50</td>
<td>4.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>3.00</td>
<td>1.63</td>
<td>2.33</td>
<td>0.94</td>
</tr>
<tr>
<td>5</td>
<td>3.25</td>
<td>1.25</td>
<td>3.50</td>
<td>1.50</td>
</tr>
<tr>
<td>6</td>
<td>3.33</td>
<td>1.70</td>
<td>3.33</td>
<td>0.94</td>
</tr>
<tr>
<td>7</td>
<td>4.00</td>
<td>1.63</td>
<td>3.33</td>
<td>1.70</td>
</tr>
<tr>
<td>8</td>
<td>3.50</td>
<td>0.50</td>
<td>3.50</td>
<td>0.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Metric 5</th>
<th>Metric 6</th>
<th>Metric 7</th>
<th>Metric 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>1</td>
<td>4.50</td>
<td>1.61</td>
<td>3.70</td>
<td>1.21</td>
</tr>
<tr>
<td>3</td>
<td>2.75</td>
<td>0.75</td>
<td>3.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>1.67</td>
<td>0.47</td>
<td>1.67</td>
<td>0.94</td>
</tr>
<tr>
<td>5</td>
<td>3.50</td>
<td>1.50</td>
<td>3.50</td>
<td>1.50</td>
</tr>
<tr>
<td>6</td>
<td>3.00</td>
<td>1.41</td>
<td>4.00</td>
<td>0.82</td>
</tr>
<tr>
<td>7</td>
<td>4.17</td>
<td>1.55</td>
<td>4.33</td>
<td>0.94</td>
</tr>
<tr>
<td>8</td>
<td>3.00</td>
<td>0.00</td>
<td>2.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

5.2 Subject Analysis

Each of the five participants in this study are considered independently in this analysis. Responses to the self-assessment questionnaire are examined first, detailing overall tendencies as well as notable trends in the responses to individual questions. Cases where a subject consistently agreed or disagreed with particular statements across all four observations sessions are also highlighted as insights into behaviors that may be prominent in the analysis of the verbal protocols. The results of the verbal protocol analysis are presented next, beginning with a discussion of the subject’s behaviors at each observation session. These are then compared over the course of the four sessions, identifying both persistent behaviors and those that changed over time. Next, the design artifact quality evaluations are introduced, studying changes in each quality metric in each of the session solution artifacts. The analysis of each research subject concludes with a summary discussion of the analysis results through a comparison of the conclusions of each analysis, highlighting the correspondences and differences between the three perspectives on the subject’s performance.
5.2.1 Subject A

Subject A was female, a junior Computer Science major who had not yet taken CSC326 (Software Engineering), and was not taking the class during her participation in this study. She had no professional experience in software development and had not taken any advanced Computer Science elective courses. Subject A worked on Design Problem 4 at session 1, Design Problem 1 at session 2, Design Problem 6 at session 3, and Design Problem 8 at session 4.

5.2.1.1 Self Assessment Questionnaire

Figure 5.1(a) shows Subject A’s aggregate scores for the novice-oriented statements in each of the three questionnaire sections over the course of the four observation sessions. These scores present a mixed view of the subject’s perception of her design skills and strategies. On one hand, the scores for all three sections show a decline in agreement with the novice-oriented statements between sessions 2 and 3. Overall, however, only the score for section 2 indicates a continued decrease. The sum of the responses for section 1 showed a sharp increase from session 1 to session 2, which was driven by the responses to question 5 that dealt with using tools to generate code as a means of developing solution models. At the first session, Subject A indicated that she strongly disagreed with using such tools to help create and maintain her view of complex design problems, while at the second and third sessions she indicated strong agreement with this statement. As shown in Figure 5.1(b), the other responses remained basically flat at the 4-5 level at all four observations, except for question 2, which alternated between “Disagree” and “Agree.”

The subject’s responses to the section 3 statements did show a decrease between sessions 2 and 3, but also showed a sharp increase at session 4. The principal driver of both of these changes are the responses to statement 28, which dealt with being closed to changing previously made decisions. At session 3, the subject indicated strong disagreement with this statement, while all 3 of the other responses indicated agreement as shown in Figure 5.1(d).

The aggregate responses for section 2 of the questionnaire seem to indicate a steady decline, but an examination of the individual statement responses leads to a different conclusion. Figure 5.1(c) shows these responses, and only statement 17 indicates a substantial change between sessions 3 and 4. This statement is about controlling the design process by relying on one’s own knowledge and experience, and at the first three sessions the subject indicated agreement with the statement. At the fourth session, the response changed to disagreement, but this is not sufficient to indicate a continuing trend, particularly considering how this subject’s responses to some of the questions alternated between agreement and disagreement over the course of the four sessions.

Subject A’s aggregate scores for the expert-oriented self-assessment questions are shown
in Figure 5.2(a). These scores suggest little change in how the subject thinks about her own design skills. Responses to the statements in sections 1 and 2 of the questionnaire showed a sharp decrease, driven by questions 3, 4, and 15, as illustrated in Figures 5.2(b) and 5.2(c). On question 3, the response changed from “Strongly Agree” to “Not Sure” between sessions 1 and 2, but returned to “Strongly Agree” at sessions 3 and 4. This question concerns the use of abstract experiences and patterns in creating and maintaining the subject’s view of a complex design problem.

The response to question 4 dropped from “Agree” to “Disagree” from session 1 to session 2 and rebounded to “Strongly Agree” for session 3. This statement is about the subject’s use of metaphors and other abstract labels to simplify aspects of a design problem and solution. This statement had the largest increase between sessions 2 and 3, and the subject remained in agreement with it through session 4, though the “Agree” response at the first session makes correlating the later increase with the exposure to the \( P^3F \) questionable.

Statement 24, with agreement indicating openness to alternatives and change as the design process progresses, also had a sharp increase between sessions 2 and 3, changing from “Not Sure”
to “strongly agree.” Like question 4, however, the subject responded “Strongly Agree” to this question at the first session. Furthermore, the subject indicated that she is generally opposed to change or modification once a design decision has been made in her response to question 28. These opposing responses prevent drawing any reliable conclusions with respect to these two statements and the behavior they imply.

This subject did show consistent agreement with several novice and expert behaviors. With regard to viewing a design problem, Subject A indicated that she is implementation-oriented (statements 7 and 10) and will try to restructure problems to match known solutions (statement 8). These self-assessments are aligned with results from other studies that identified similar tendencies among novice, inexperienced designers. Subject A also showed consistent agreement with statements 14 and 16, indicating a desire to concentrate on only one issue or element at a time and to have access to complete, concrete information about the problem. These responses are also aligned with prior results that suggest novice designers have a narrow and literal view
of the problem as stated. In the questionnaire section concerning how design decisions are made, Subject A’s responses suggest that she makes decisions based strongly on the explicitly stated requirements, particularly as she understands them (statements 22 and 26). This type of behavior has also been noted in other research. Subject A did not show a consistent disagreement with any of the novice-oriented statements on the questionnaire.

Examining the questions where agreement implies an orientation towards an expert-like behavior, Subject A consistently agreed with three statements. Responding “Agree” in all four sessions to question 11 would suggest that this subject maintains a flexible and opportunistic design process. Responses to question 18 assert that she regularly reviews the work in progress. Finally, Subject A indicated “Strongly Agree” to question 32 at all four sessions, suggesting an orientation towards the overall purposes and goals for a system. The subject’s responses to other questions call these three affirmations into question, particularly question 32. As noted earlier, this subject indicated a strong reliance on concrete information and explicitly-stated requirements, but overall organizational goals and purposes are rarely included in a requirements document. Subject A also consistently disagreed with two expert-oriented statements (13 and 27), suggesting that she does not have a high tolerance for uncertainty and does not like to use “soft” information in her design decision-making process. These responses are related to each other and also reflect Subject A’s need for concrete information and hard facts as indicated by her responses to questions 14, 16, 22, and 26 discussed above.

Overall, the self assessment data for this subject do not support drawing a causative link between the introduction of the $P^3\mathcal{F}$ and self-perceptions of design skills and strategies. The subjects’ responses do, however, corroborate earlier research and provide insights into behavioral traits that may be evident in the protocol analysis.

5.2.1.2 Verbal Protocol

At the first observation session, Subject A worked on Design Problem 4, the investigation service case management system. The subject’s initial focus was on hardware requirements, particularly the amount of storage space needed to hold the company’s information. Her attention then abruptly turned to the different kinds of cases the company investigates, attempting to develop a class hierarchy directly from the problem description and assumptions based on her own knowledge. The only questions asked to this point were about the storage requirements. The subject’s focus turned abruptly again, this time addressing the kinds of information that would need to be stored. In this case, she did ask questions, seeking specific details about the different types of files to be stored. This information-gathering episode spanned a sequence of twenty-seven consecutive coded design decisions, the longest series of decisions on a single aspect of the system during this session.
Once she felt that she had enough information about the information to be store in the new system, and without any indication that her focus was going to change, she made a sweeping storage and user interface proposal. She continued detailing this proposal for 3 minutes and 18 seconds before pausing to ask the “customer” if this sounded like a good solution. When asked about the security of this proposal, she immediately changed her mind and suggested a more secure storage solution. Again she applied her own experiences, proposing a Facebook or MySpace style of interface for users. The subject spent a sequence of thirteen proposal-type design decisions explaining how this profile-based interface would work. Again, her attention shifted suddenly, prefaced by the comment “this is just some random thinking,” to archival data storage. After a brief consideration of a small set of backup and archive options, the subject’s attention changed direction to gathering information about the different types of users who would use the system and their roles. This led her to reconsider her earlier decision to base the overall database structure on cases and instead root it in the different types of users. The session ended with an attempt to refine the system structure based on this new perspective.

Throughout this observation session, Subject A exhibited many behaviors that have been associated with novice designers. She relied heavily on personal experience and the information explicitly given in the problem statement to construct her solution proposals. Her approach did not seem to have an overarching plan or order as she skipped around through the problem space in a trial-and-error manner, trying to force the problem to fit familiar solutions. One significant problem she had was with recognizing relationships between the different entities in the design problem, for example, how different users interacted with investigation cases and their associated information sources. Additionally, she seemed to have difficulty in understanding the legal and business contexts in which the system would have to operate. Subject A did revisit previous decisions and proposals, although this seemed to be a product of realizing she might have made an error rather than a planned, iterative refinement process. Overall, this subject’s behavior during this observation was strongly novice-like.

Subject A worked on Design Problem 1, the payroll system, during the second observation session. In contrast to the first session, the subject began this session by immediately proposing the construction of a Unified Modeling Language (UML) diagram to represent her solution proposal. Her initial focus was to generate all of the details necessary to describe an employee, drilling down to specifics such as the lengths of the character strings necessary to represent the various employee attributes. This line of thought continued with a detailed explanation of how different class objects in the system would calculate various deductions and generate paychecks. This, in turn, led to a series of proposals concerning different types of employees based on their pay schedules and rates. At this point, her attention shifted from the decomposition of employee characteristics to the sequence of steps necessary to process paychecks. Subject A then started constructing a finite state machine (FSM) to document this process, and noted that “there so
many things that are happening, it’s going to be huge” with concerns about the complexity of the process. Sixteen proposal-type design decisions were spent describing, in detail, the payroll process as she understood it from the problem statement, while documenting the process in a FSM. When the subject completed her FSM, her focus abruptly changed to the hardware environment and requirements, where the session ended.

Subject A’s approach to this problem was very much in line with the behaviors identified in other studies of novice designers. In particular, she exhibited very narrow focus and a linear decomposition approach to drilling down from the stated requirements to very specific details on system entities. She did not generate or consider alternatives, instead relying on her own knowledge and the problem statement for nearly all of her information. Beyond the high-level goals of creating UML and FSM diagrams to document her design, Subject A did not appear to have a coherent plan for designing the system other than to pick a particular entity or object in the system and dig down to discover as much detail as possible about it. Overall, Subject A’s behavior during this session was consistently novice-like based on the behaviors and strategies identified in prior research.

At the third observation session, Subject A chose to work on Design Problem 6, the nursery information systems problem. In contrast to the previous sessions, the subject began by gathering information about the business context of the problem: where the company was located, where the customers were located, and issues related to interstate commerce. Once she had a clear view of the business environment, she expressed a high-level plan for solving the problem. The subject recognized two areas of concern within the overall development problem. First, the company needed a more robust and flexible inventory management system. Second, there were many shortcomings and inefficiencies in the current billing and customer management system. Subject A chose to gather information about customer information first. She approached this by asking questions to determine different types of customers and their important characteristics, but did not seek to identify low-level details. As the subject began documenting her design, she recognized that the billing, customer information, and inventory systems would need to be parallel within the new system rather than tightly intertwined.

Subject A’s focus widened slightly to begin relating key customer information, such as credit information and outstanding invoices, to how different types of customers would be billed for their purchases. This also encompassed differences in invoicing for in-state and out-of-state customers that needed to be automated in the new system. As she gathered information, the subject began making tentative solution proposals, but recognized that these were not final decisions. Information gathered about invoice structure led to a brief investigation of some details about inventory items, but only as they applied to the invoicing and billing processes. Understanding the parallel nature that was evolving in her system design, Subject A identified the need for representing the employees who maintain the inventory, interact with customers,
and perform the various billing-related tasks. After gathering some details about different types of employees, her attention turned back to the inventory system, this time with the goal of identifying how the inventory records are updated based on customer purchases.

Subject A’s information-gathering episode dealing with inventory management began with a customer-centric approach, identifying the differences in the purchasing processes for on-site and phone-based purchases. With this information in hand, the subject began integrating all that she had learned into a high-level design for the system. She maintained the parallel structure started earlier in the design process, and filled in new details that had been gathered. More importantly, she also identified and described relationships between the different parts of the proposed system and between different entities within those parts. Subject A incorporated the invoicing and billing processes into the inventory management to accomplish near real-time inventory updating as described in the problem’s requirements. Throughout her description of her design proposal, she was careful to maintain both the integration of the various parts of the system and a separation of those parts as distinct entities within the system.

While Subject A did continue to exhibit novice behaviors such as relying on her own experiences and digging for specific details on some entities within the system, she also showed signs of more expert-like behavior. In particular, she seemed to take a more organized approach to solving the problem rather than a random, trial-and-error path. She also tried to establish and maintain relationships between the various parts of her proposed solution. As new information became available, the subject also revisited earlier decisions where the new knowledge was applicable. In contrast to her behavior during the first two sessions, she attempted to maintain a high-level view of the system and how different users would interact with it. Overall, Subject A demonstrated a significant move towards more expert-like behavior during this observation session.

During the fourth observation session, Subject A worked on developing a solution to the restaurant delivery service system described in Design Problem 8. Much like her approach in the third observation session, the subject began by gathering information about the business process and where the new system would fit within that process. In addition to the direct users of the system, she also directed attention to non-users that could be impacted by the implementation, in particular, the restaurants that have arranged to use the delivery service. She ended this initial information-gathering sequence with a review of what had been learned and giving it some structure. As she was doing this review, the subject also noted several pieces of information that were not useful at this time and did not incorporate them into her structured view.

With this basic understanding of the business processes, Subject A began refining her view, asking questions to fill in details and incorporating that information into the appropriate areas of her outline. There were two interesting aspects of this information-gathering episode. First, she did not try to drill down to low-level details at once, but took a breadth-wise approach to
gather new information about different entities of the process at the same level of detail. Second, her primary focus was on the relationships between the various parts of the system and pieces of information rather than on the details themselves. This iterative, breadth-wise process of discovery and refinement came to characterize the remainder of the observation session.

Subject A did get bogged down at one point, spending nearly 20 minutes exploring different methods of acquiring menu information from the restaurants and integrating it into the new system. During this episode, she did use a trial-and-error approach, proposing a solution then evaluating it through a discussion with the customer. However, at this point in her design of the system understanding how this information would be gathered and used was critical to making progress towards completion. Also, she did stay at a relatively abstract level and did not try to make design decisions dealing with low-level details that would not be important until a satisfactory conceptual solution was determined. Once that solution was found, she moved on to other aspects of the system design.

Overall, this session represented a further move towards expert-like design behavior, particularly in her search patterns through the problem and solution spaces. Although she did not explicitly express a high-level plan for solving the problem as she did at the start of the third session, her progress throughout this session was organized in a way that reflected an internal plan of action. Subject A’s use of a breadth-first information gathering and solution proposing process demonstrated an iterative and agile-like approach to solving the problem. She also exhibited the ability to discern relevant versus irrelevant information based on the current context she was working in.

Reviewing all four sessions for Subject A, the behaviors she exhibited provide evidence that the introduction of the $\mathcal{P}3\mathcal{F}$ helped her begin to develop expert-like design strategies. Her behavior and thought processes demonstrated in the third and fourth sessions were dramatically different than in the first two sessions. While she did not explicitly use terminology incorporated in the framework, she did apply the conceptual processes. In particular, the third and fourth sessions demonstrated a much more organized and thoughtful approach to the problems. She was aware of and actively interacting with the larger context where the completed system would operate. The subject also seemed much more confident in her decision-making, perhaps because she was not trying to make detailed decisions with insufficient information, but committed only enough to keep the design process moving.

5.2.1.3 Design Artifact Quality

Overall, Subject A produced low-quality solutions to the design problems, based on the independent evaluations shown in Figure 5.3. In particular, her evaluation scores for observation session 1 (problem 4) was uniformly “very poor,” with one of the evaluators commenting that
the designer “obviously did not understand the building blocks of the system.” Two other subjects chose this problem to work on and this subject’s evaluations were below average for all eight of the rubric metrics. For this subject, the results of the design quality evaluation do not provide any evidence that the introduction of the $\mathcal{P}^3F$ had any positive effect on her design performance.

Subject A did produce a better solution for Design Problem 1 during session 2. Only two of the metrics were scored below average for this problem: Creativity and Scalability. One of the evaluators noted that the solution “meets customer requirements and general technical criteria” for the problem, and that it was “not creative but logical.” After the introduction to the $\mathcal{P}^3F$ during the second session, the evaluations for the third session (problem 6) dropped significantly, with all eight metrics evaluating below average. Evaluator comments on this solution suggest confusion on the part of the subject: “fundamental disconnects with issues to be addressed and requirements to be incorporated,” “fails to identify key entities and relationships,” and “little critical thinking.” The evaluations for the fourth problem were also poor, reflecting the evaluators’ opinions that the solution was “superficial and incomplete” with little attention to the data requirements or technical criteria given in the problem statement.

Nothing in this data suggests a performance improvement resulting from the exposure to the $\mathcal{P}^3F$. Subject A’s best evaluations were on the problem worked during the second observation session, before the introduction of the framework. It should be noted, however, that this subject did not have any professional software development experience, and had not yet taken the undergraduate software engineering course. Her exposure to software design problems of the

Figure 5.3: Subject A Design quality evaluations
scope and complexity of those used in this study prior to her participation was minimal at best, and this could be a contributing factor in her overall poor performance.

5.2.1.4 Subject A Summary

Subject A’s self-assessments and design quality evaluations did not indicate any effects that might be attributable to the introduction of the $P^3F$, although the analysis of the verbal protocols suggests there might be some small effect reflected in the subject’s observed behaviors, as noted above. Although she continued to rely on the explicit information given in the problem statement, she also seemed much more willing to ask questions to gather additional information after the introduction of the framework. In the third and fourth observation sessions, Subject A was also more organized in her approach to the design problems and tried to construct and maintain a big-picture view of the problems and solutions. The $P^3F$ process is stated in a step-by-step fashion and incorporates steps that ask the designer to consider the context and environment of design decisions, and this construction was intended to make it easily accessible and applicable for novice users. While Subject A did not follow the process in an explicitly observable manner during the last two sessions, the iterative revisiting of her view of the system as a whole and a more organized strategy for approaching the problem suggest that she was trying to use the process in her work on the problems.

Comparing the self-assessment results with the observed behaviors also revealed similarities between Subject A’s perceptions of her design strategies and the strategies she used while working on a design problem. In her self assessments, the subject consistently agreed with several statements indicating novice-like design thinking. The responses to these statements suggest that the subject considered herself to be implementation oriented, tries to fit problems to known solutions, desires explicit and concrete information, and relies on personal experience. These behaviors were very evident during the first two observation sessions, corroborating the self-assessment results. During the last two sessions, these behaviors were still present but to a lesser degree than in the first two sessions, although this was not reflected in the self-assessment responses. This suggests that she was consciously trying to use aspects of the $P^3F$, but that she did not have enough experience with the framework to internalize it to the point of altering her self perceptions.

The artifacts Subject A produced for evaluation were not consistent with the verbal descriptions made while she was constructing the artifacts. In particular, the subject verbally described her solution in clearer detail and with better organization than what she put on paper. This inconsistency remained constant over all four sessions and the resulting artifacts, suggesting that this subject might be better at describing what she is thinking than she is in documenting it in some explicit form, particularly when time is limited.
Overall, the results for this subject suggest a modest move towards more expert-like behavior when designing solutions to complex software system problems. The observed change may have been due only to a conscious effort to apply the most accessible part of the $\mathcal{P}^3\mathcal{F}$ for the benefit of the study, but because the subject did not make specific references to the framework it is difficult to determine if that was indeed the case. The observed behaviors also confirmed several aspects of Subject A’s novice-oriented view of herself.

5.2.2 Subject B

Subject B was male, a junior Computer Science major who had not yet taken CSC326 (Software Engineering), and was not taking the class during his participation in this study. He had no professional experience in software development and had not taken any advanced Computer Science elective courses. At the first observation session, Subject B worked on Design Problem 7, followed by problems 3, 6, and 1 at the second, third, and fourth sessions, respectively.

5.2.2.1 Self Assessment Questionnaire

Figure 5.4(a) presents the aggregate scores for Subject B’s responses to the novice-oriented statements in each of the three questionnaire sections over the four observation sessions. All three possibilities are represented in the interval between observation sessions 2 and 3: the section 1 aggregate score decreased, the section 2 score remained flat, and the section 3 score increased sharply. Based only on these summations, it might seem that the introduction of the $\mathcal{P}^3\mathcal{F}$ induced an increase in novice-oriented attitudes. Responses to four questions drove this change, and in all four cases the level of agreement did not continue to increase with the session 4 responses. The aggregate score for section 2, flat between sessions 2 and 3, does show a decrease in agreement at the fourth observation session. However, the aggregate score remained higher than the sum of the responses at the first session. The total score for section 1 of the questionnaire indicates a continued decrease in agreement with this set of novice-like behaviors, but the responses to the individual questions do not support a correlation with the introduction of the $\mathcal{P}^3\mathcal{F}$.

Responses to the questions in section 1 are shown in Figure 5.4(b). The aspect of these responses that stand out the most are the consistent agreement with questions 2 and 8. Agreement with question 2 indicates that Subject B maintains a narrow view of a system he is working on, while the strong agreement with statement 8 demonstrates a desire to reorganize a problem to fit with known solutions. Overall, Subject B’s responses to this set of question does not provide any evidence for a change resulting from the introduction of the $\mathcal{P}^3\mathcal{F}$. The same can be said of his responses to the section 2 questions as shown in Figure 5.4(c). None the responses in this section changed between sessions 2 and 3.
As noted above, the aggregate score for section 3 sharply increased after the introduction of the $P^3F$, driven by responses to four questions. The responses to these individual questions, shown in Figure 5.4(d), show an increase in his level of agreement or decrease in level of disagreement, but do not continue at that same rate of change in the session 4 questionnaire.

Subject B’s aggregate responses to the expert questions in the self-assessment questionnaires were mixed as Figure 5.5(a) illustrates. The scores for section 1 were basically constant over the four sessions, while the section 2 and 3 scores showed a sharp decline and rise after the introduction of the $P^3F$ during session 2. The decrease in the section 2 aggregate is not driven by any particular single question responses. However, the section 3 increase between sessions 2 and 3 is largely a result of 2-level increases in the responses to questions 30 and 32. At the second session, the subject signified strong disagreement with statement 30, suggesting he did not consider his design decision process to be creative. This response changed to “Not Sure” in the response to this question at sessions 3 and 4, possibly indicating a reconsideration of this
aspect of his decision-making. Subject B’s responses to question 32, dealing with an orientation towards the overall purposes and goals of a system as a decision-making motivation, went from “Disagree” to “Agree” after the introduction of the $P^3F$.

Between these two questions, only the responses to question 30 seem to be significant. As shown in Figure 5.5(d), the subject’s response to question 32 at the first session was “Agree” before dropping to “Disagree” at the second session and returning to the first response at the last two sessions. On question 30, however, Subject B responded “Disagree” at the first session before indicating “Strongly Disagree” at the second session. Overall, the subject’s self-assessment of this decision-making aspect transitioned from disagreement to uncertainty, the only clear movement towards expert-like self-perception in this subject’s questionnaire responses.

Questions 13 and 15, on the other hand, suggest a decrease in expert-oriented behavior. The aggregate scores for these questions are shown in Figure 5.5(c). Although the responses to these two questions show an overall decline in agreement with expert-like thinking, a connection with the introduction of the $P^3F$ is not clear from the series of responses. Subject B’s largest decline, from “Agree” to “Disagree” on question 13, occurred between sessions 1 and 2, before the $P^3F$ was presented. The responses to question 15 also dropped from “Agree” to “Disagree” but did so in two steps from session 2 to session 4. However, at the first session the subject indicated “Disagree” to this question, the same response as he gave at the final session.

Subject B did show consistent agreement with several expert and novice behaviors and strategies. With respect to viewing the design problem, the subject indicated agreement with the novice-like behaviors in questions 2 and 8 (Figure 5.4(b), maintaining a narrow focus on the problem at hand and trying to restructure the problem to fit familiar solutions, respectively. These behaviors have been previously associated with novices. The subject also indicated agreement with the expert-like behaviors involving the use of abstractions, patterns, and mental simulation models, questions 3 and 9 in Figure 5.5(b).

In section 2 of the questionnaire, dealing with the manipulation and navigation of the design space, Subject B signified consistent agreement with addressing one issue at a time (question 14) and relying on personal experience to control his design process (question 17), as shown in Figure 5.4(c). In contrast to this, he asserts that his design process is flexible and opportunistic (question 11) and that he also takes a critically reflective approach to reviewing his work (question 18).

Subject B’s responses to the questions in section 3 of the questionnaire, shown in Figure 5.5(d), suggest an orientation towards the use of explicitly stated information (question 22) and personal experience (question 23) in decision-making. He feels confident in his decisions (question 21) and is willing to revisit and change decisions that have already been made when necessary (questions 24 and 28). The contrast of these novice-like and expert-like behaviors is not typical, and will be behaviors of interest in the verbal protocol analysis.
5.2.2.2 Verbal Protocol

At the first observation session, Subject B chose to work on Design Problem 7, the restaurant information systems problem. Relying solely on the problem statement and his own experience, the subject immediately began trying to develop a database structure. He included very specific details such as the ingredients for various menu items and the data types of different fields in the database. His attention then shifted to hardware requirements such as the kind of computer needed for the database server and touch-screen systems for the cash registers.

His focus shifted abruptly again, this time to a high-level description of the process his system would use. His documentation of this workflow was through an ordered list of steps. He chose not to use any kind of modeling notation like a finite state machine or UML sequence diagram. The subject also proposed, without connection to any prior discussion, adding an automatic stock ordering system which was not identified as a requirement in the problem statement. His rationale for this proposal: “because people are lazy and it’s better if computers can do the work for them.” After a brief discussion with the customer, he reluctantly decided to
just provide a daily inventory report that could be reconciled manually against the stock on hand.

Another abrupt shift occurred after this exchange. The subject proposed integrating the company’s financial system with his inventory and point-of-sale system because his system was already going to be keeping track of orders and the money collected as well as the inventory and its associated cost. Attention then turned to the interaction of the cash register systems with the database and an order display system in the kitchen area. The only issue addressed in detail in this part of the system proposal was a means of queueing and tracking the orders as they are placed, sent to the kitchen, and completed. This led to gathering information about reconciling the cash in the registers and the impact of this process on the database and reporting parts of the system. The reference to register sales reporting triggered several more proposals about different types of reports that the system would provide.

Subject B’s attention then switched to hardware requirements again, this time focused on touch-screen and credit/debit card readers and PIN pads. He then returned to give more details about the reporting subsystem, proposing both scheduled and on-demand reports in addition to proving the ability to define and save customized reports. When asked to describe the computational structure of his proposal, the subject seemed confused and resistant. He said he could provide a drawing of the hardware interconnections, but the software structure was too complex to put into any kind of diagram. When asked if he would like to meet with the cashiers or kitchen staff to get their input on the user interfaces, he said that would not be necessary since “your employees would probably like the new system better.”

Expressing some unease with the subject’s resistance to providing additional documentation of his proposal, the customer asked him why he should be chosen to implement the system over other bidders. Subject B’s response: “I can do it cheaper and I can do it faster.” The subject concluded his system proposal with a description of how his system would record and report cost of materials for the food items sold. This description was somewhat shallow, and did not account for waste, minor variances in the amount of materials used in various menu items, or the variability in food prices.

The design strategies demonstrated by Subject B during this session were very novice-like. He did not seem to be able to maintain focus on a single aspect of the problem or his proposed solution for long, and his treatment was very superficial. He projected an attitude of “trust me, I’m the programmer and I know what is best for your needs.” He proposed many features that were not included in the problem statement because he felt that they were good ideas. He did not seem interested at all in understanding the company’s particular business processes and relied on his own observations and experiences with other restaurants. His overarching approach could be described as trial-and-error. When topics were revisited, he proposed something different, but it was not clear that this was a result of evaluating the previous decision or just a proposal.
of something different.

Subject B chose to work on the knowledge management system, Design Problem 3, during his second observation session. As he did during the first session, the subject started by immediately trying to develop a database structure, and again defining data fields very specifically, down to the level of data type and length. He did not ask any questions about the company’s existing systems or business processes. Without asking if this was what was desired, he began describing his proposed web-based system, noting that “MySQL and Apache run on everything anyway.” His proposal went into considerable detail, both in terms of the user interface and the interactions between the web forms and the database system. He did not seem to be interested in any more than a superficial login for access control, and did not address issues related to what information the users could enter or modify in the system.

During this time, the subject was constructing what he called a “tree diagram” that was supposed to map out the database tables as well as the pages and forms of the web interface to the system. When asked about the workflow of the system, Subject B explained that there would be a “main” page from which the other pages could be accessed. The customer expressed concerns about how this new system would fit in with the company’s normal workflow and business processes. The subject replied “There’s not much information about that.” The customer provided a brief and high-level overview of the work process, from the initial client contact through the end of the contracted work. The subject immediately proposed several modifications and additions to his existing database structure and web interface without seeking any additional information.

The subject became confused by the relationships between the consultants, the clients, and the company, particularly when an assignment had more than one consultant providing services for a particular client and contract. His confusion increased when he was told that different roles within an assignment could have different priorities with respect to that particular assignment and that different consultants could work on a particular assignment at different times. This generated a sequence of decisions proposing modifications and additions to the database structure to accommodate these new requirements. Subject B followed this with a sequence of proposal decisions describing the report generator he planned to incorporate into the new system and the variety of reports that it would produce. He did not ask what reports would be needed or whether the proposed reports would meet the customer’s needs.

When the subject exhausted all of his ideas about the information storage and access requirements for the new system, his attention turned to the hardware requirements, specifically, the location of the database server that he viewed as the main part of the new system. Subject B’s proposal was for a central database and web server that would host the system application and data. When the customer expressed concerns about “putting all of our eggs in one basket,” the subject proposed a simple mirroring process and stuck with this proposal even after the
customer stated further concerns with this arrangement. At this point, the subject then stated “sorry, I lied earlier,” explaining that a distributed network of primary and mirroring servers would be a better solution, although he did not explain why.

Subject B’s behavior during this observation session was very self-centered, as evidenced by his continued efforts to fit the problem into a familiar solution. The subject spent long sequences of decisions formulating and proposing specific aspects of the completed system, relying almost entirely on the explicitly stated requirements in the problem statement and on his own experience and knowledge. His technique for moving through the development process was not quite trial and error, but was haphazard and did not seem to use a predefined method for choosing what aspect of the problem to work on next. Instead, he made progress in fits and starts, intertwined with pauses where he “did not know what to work on next.” Overall, he did not exhibit any expert-like behaviors but did perform in a very novice-like manner.

At the third observation session, Subject B chose to work in Design Problem 6, the nursery information system problem. After reading through the problem, he noted that “this is just another database with a Web interface kind of thing” and began proposing specific details of the database structure through a sequence of eleven coded proposal decisions. The subject’s attention shifted to the customer billing process, relying on the database structure he had proposed as his primary source of information. When the customer called his attention to the requirements related to changes in the billing cycles and structure in the new system, Subject B became confused about the differences and how the customers would choose which billing method to use. After a sequence of five information-gathering (question and answer) decisions, the subject had enough information to propose a solution to this aspect of the billing problem.

The subject continued to have confusion and misunderstandings about the billing process, however, and proposed several “hackish” solutions until one was found that met the customer’s approval. At this point, about fifteen minutes into the session, the subject stated: “I’ve covered everything I’ve read about here,” indicating that he felt he had completely solved the problem. When the customer noted several key requirements that had not been addressed, the subject began a superficial description of how those requirements were met in his solution. He described how small, specific parts of the system would work independently without any explanation of how those parts would function together as part of the larger business processes the new system was supposed to help manage.

Subject B then stated that he had added various new tables to the database and described the contents of those structures. These tables were proposed to integrate the newly described processes into the overall structure of the system, but no explanation was offered about how this would happen. The subject also began to propose a variety of new processes that “the computer will perform.” Several of these proposals were not in the problem statement, but the subject said that “they will be helpful in the long run, you’ll see.” Seven episodes of new feature proposal,
customer questioning and subject responding, and either abandoning or accepting the proposal occurred before the subject decided that he could not come up with any more new ideas. When asked to document his proposed system solution, the subject began creating a bullet list of the features and processes he had proposed. As his development of the list progressed, Subject B noted that he had forgotten to include fields and a data table into his database structure to handle access permissions. This resulted in a series of fourteen propose and question decision sequences before the subject felt like he understood this implicit requirement and had proposed an adequate solution.

Throughout this session, Subject B relied heavily on his own programming experiences and tried very hard to make this new problem fit within the scope of those experiences, a documented novice trait. He made little effort to discover and understand the relationships that might be present between different parts of the new system or to understand the business processes that the new system is supposed to improve. Many times throughout the session he made proposals outside the scope of the problem statement, based on his opinion that they were the best thing to do for the company. Overall, this subject did not show any significant behavioral changes from the first two sessions, suggesting that the introduction of the $P^3F$ did not have an immediate effect on his design behavior.

Subject B chose to work on Design Problem 1, the payroll system, at the fourth and final observation session. He began this session in much the same manner as the previous three sessions, by starting to construct a scheme for the database at the core of his proposed solution. However, at this session the subject began asking questions about how the employees record their hours worked and how their gross pay was calculated. This was a notable change from the first three sessions, where the subject basically said that his way was the way it was going to be. In this question-and-answer sequence, the subject made nineteen information-gathering decisions (questions asked and answered) and six specific design proposals. Also different than the first three sessions was the depth to which Subject B specified elements in his database design. Rather than indicating field lengths and data types, the subject only used labels to map data elements to concepts in the application domain.

The subject next turned his attention to how union dues are collected from employees and paid to the different unions representing workers. In this episode of the session, the subject continued to ask questions to gather information before making proposals. Once he established his understanding, he began making solution proposals related to this part of the new system. Many of these proposals were made less confidently than in the prior sessions, at least on the surface. Proposals were often phrased as questions rather than assertions. However, the subject’s proposals were thought out more completely than in the previous sessions, and there appeared to be a logical ordering driving Subject B’s design process even though he had not mentioned a planned approach to the problem. The subject also identified and resolved several critical issues
that were not documented in the problem statement, such as what to do if the union deductions exceeded the after-tax net pay for a pay period. This led to a series of questions about how taxes are calculated and deducted from an employee’s gross pay. Within this sequence of decisions, the subject asked eight questions and proposed six abstract modifications to the database structure to accommodate calculating and recording tax-related deductions.

Throughout most of the session, the subject was documenting his design using a nested list kind of word-processing document. He explained that he “did not know how to use any of that diagramming software,” and that hand-written drawings get too messy to be useful. After working through the structural aspects of the new system, Subject B began describing the user interfaces he could create for the new system. In this area, he fell back into old habits, asking very few questions and basically telling the customer what he was going to use. He was not interested in referencing any of the current forms or spreadsheets currently being used to perform the payroll tasks. Interspersed among his description of the interfaces, he noted that “the computer...” or “the automatic program...” would perform all of the functionality needed, only displaying the results of the computations to the user when necessary. The final topic covered by the subject’s proposal was a few additions to the database structure to handle access control permissions for the payroll system.

During the fourth observation session, Subject B did have a few episodes of more expert-like behavior than in the first three sessions. In particular, he asked more questions and actually seemed to listen to the answers in order to integrate them into his solution proposals. Also, he seemed to take a more organized approach to investigating the problem and developing a solution. Subject B did not try to dig down to concrete, implementation-level details. However, many aspects of his behavior were still novice-like, especially when it came to the specification of user interface designs. He clearly assumed that he knew what was best, and was not interested in learning about the company’s workflow and business processes.

Considering Subject B’s behaviors across all four observation sessions, he demonstrated minor changes after the introduction of the $P^3F$. These changes, noted only in the fourth session, included a little more attention to the customer’s actual needs, at least through the first half of the session, and a less random and haphazard approach to developing a solution. Whether or not these changes can be attributed to the introduction of the $P^3F$ is difficult to determine, primarily because these changes did not occur immediately with the experimental treatment and because the differences in behavior were relatively small.

5.2.2.3 Design Artifact Quality

Subject B did demonstrate performance improvements between sessions 2 and 3, based on the design quality evaluations for those two problems, as shown in Figure 5.6. However, the
evaluations of the solution produced during the second observation session were the lowest of the four problems he worked on, and represented a drop in at least two evaluation points for all but one of the metrics in the rubric from the scores on the first problem. Additionally, the evaluations for Design Problem 7, completed during the first observation session, were substantially above average for the problem, making the drop at the second session (problem 3) more significant. The evaluators’ comments highlight their perception of the performance change, noting “comprehensive understanding addressing each of the affected areas” at the first session, but “barely captures fundamentals” at the second.

Figure 5.6: Subject B Design quality evaluations

The evaluation scores for the third session problem improved across the board, with the largest change from session 2 occurring in the Creativity metric. The evaluators noted that this solution represented a “very good holistic view” and that it was “likely to exceed expectations.” However, Subject B’s performance did not continue to improve consistently with his solution to Design Problem 1 at the fourth session. Both evaluators noted a lack of attention to technical details given in the problem statement as a reason for lowered evaluation scores.

Overall, the improvement shown between sessions 2 and 3 in the artifact evaluations could be a result of the subject’s introduction to the $P^3F$, but the data do not strongly support that conclusion. The weak point is the decline between sessions 1 and 2. Subject B’s performance on the first problem is well above average for all of the metrics, while it is close to the average for the session 2 problem. The subject’s lack of experience with complex design problems could also be a factor since he has not yet taken Software Engineering and has no professional work experience in software design or development.
5.2.2.4 Subject B Summary

None of the results accumulated for Subject B suggest that the introduction of the $P^3 F$ had any significant effect on the subject’s design strategies. His self-assessment responses indicated a consistent novice-like view of himself, including characteristics such as a narrow focus, the desire to restructure problem to fit with known solutions, a strong reliance on personal experience and explicit information. The subject also responded that he perceives himself to be a confident designer, willing to revisit and change decisions made earlier in a design process. The results of the verbal protocol analyses and the design quality evaluations suggest that his strategies are more novice-oriented than expert-like.

During all four of the observation sessions, Subject B relied heavily on his own experiences and on concrete details explicitly stated in the problem description. He also worked hard to restructure the problem to fit a solution he was familiar with, regardless of whether or not this solution was the best one for the situation. The subject’s strong focus on a single issue at a time precluded the development of an integrated view of the system.

Subject B was confident in his decisions, but this confidence was often manifested in an arrogant or domineering manner. The subject’s confidence rested on his apparent presumption that because he was the “expert” in the conversation, he knew what was best, and his proposals should not be questioned. He was also willing to change directions at will while designing, but his behavior during the observations seemed haphazard and seemed to employ a trial-and-error approach. The lack of serious and sustained attention to any part of the problem resulted in shallow and superficial solutions that demonstrated a lack of attention to detail.

The subject’s behaviors, solution quality, and self-assessment analyses do not support a conclusion that the introduction of the $P^3 F$ triggered any changes in Subject B. The subject was not observed to exhibit an increase in expert-like behaviors or a decrease in novice characteristics, and his perceptions of his own design skills and strategies did not change in a significant way. Furthermore, while the quality of the solution produced for the third problem was a significant improvement over the evaluation of the second-session artifact, the scores for the second solution were much lower than the first. Only two of the third artifact metrics scored higher than the first session, so the increase in quality from session two to session three was more like a recovery to the initial quality level for most of the metrics.

5.2.3 Subject C

Subject C was a male second-year graduate student in the Master of Computer Science program. He had one year of work experience in software development, and had taken many advanced computer science courses at the undergraduate and graduate levels. Subject C worked on Design Problem 1 at session 1, Design Problem 4 at session 2, Design Problem 8 at session 3, and
Design Problem 5 at session 4.

5.2.3.1 Self Assessment Questionnaire

Subject C’s aggregate scores on the novice-oriented questions on the self-assessment questionnaire, shown in Figure 5.7(a), present a mixed view of his perceptions of his design strategies and behaviors. The sharp decrease in agreement with the questions in section 1 was driven by 3 point drops in the responses to questions 2 and 10. The contour of the plot for the section 2 scores is primarily a function of the responses to question 20, which decreased 3 points between session 1 and 2, followed by a 3 point increase between the responses for sessions 3 and 4. The response changes for questions 26 and 31 are responsible for the aggregate score valley at session 2.

Subject C’s session 1 and 2 “Agree” responses to question 2 suggest he felt that he maintained a narrow view of the system focused only on what he was working on at the time. As shown in Figure 5.7(b), his self-assessment of this behavior changed to “Strongly Disagree” and continued at “Disagree” at session 4 after the introduction of the \( P^3F \). Similarly, he responded “Strongly Agree” to question 10 at the first two sessions, followed by “Disagree” at session 3 and “Not Sure” at session 4. These responses suggest an effort to develop a broader view of the system under consideration, a less novice-like behavior.

The responses to the novice-oriented questions in section 2 of the questionnaire, shown in Figure 5.7(c), do not indicate any obvious relationship with the introduction of the \( P^3F \), with three of the questions having constant “Strongly Agree” responses across all four sessions. The two questions with large declines in agreement, numbers 17 and 20, both show this change occurring between the first and second sessions, prior to the introduction of the \( P^3F \). While the consistency of the last three responses to question 17 suggest disagreement with the idea of using one’s own knowledge and experience to control the individual design process, because the change occurred before the subject was introduced to the \( P^3F \), it is unlikely that this attitudinal change is a result of the \( P^3F \) because the decrease did not coincide with the introduction of the \( P^3F \).

The same argument applies to Subject C’s responses to the novice-oriented questions in section 3 presented in Figure 5.7(d). Three questions (26, 28, and 31) had 2 point decreases between the responses at sessions 1 and 2. In the case of questions 26 and 31, the subject’s responses returned to and remained at the session 1 level at the last two observations.

Subject C’s aggregate scores for the expert-like statements, shown in Figure 5.8(a), all indicated agreement increasing between sessions 2 and 3, but the responses to the section 3 questions would seem to be the only ones that could possibly be associated with the introduction of the \( P^3F \). The scores for sections 1 and 2 also increased between the first and second sessions.
The rate of increase for the section 1 responses was slightly less between sessions 2 and 3 than sessions 1 and 2, while the rate of change remained constant for the section 2 score. Examining the responses to individual questions presents a different view.

Among the expert-oriented questions in section 1, shown in Figure 5.8(b), the largest response change to an individual question occurred between sessions 1 and 2 on question 6, with Subject C’s response changing from “Disagree” at session 1 to “Strongly Agree” at the other three sessions. This could indicate an increased tendency to create multi-level models while designing software or a poorly-thought response at the first session. On question 9, however, the subject responded “Not Sure” at the first and second observations and changing to “Strongly Agree” at the third and fourth. Since this change coincided with the introduction of the $P^3F$, it might be a result of this intervention.

Among Subject C’s responses to the section 2 questions, only question 13 is notable at all. Agreement with this statement indicates a strong tolerance for uncertainty and ambiguity in the subject’s design thinking. His responses began with “Strongly Disagree” at session 1, jumped to “Not Sure” at session 2, another jump to “Strongly Agree” at session 3, and a drop back to
“Not Sure” at session 4, as shown in Figure 5.8(c). Two aspects of this response curve suggest it was not a result of the $P^3F$ introduction: the same rate of change across the entire session 1 to session 3 interval, and the subsequent decline in agreement that occurred at the fourth session. The other questions in this section do not provide evidence of any remarkable change over the course of the study.

Three questions in section 3 might indicate a response due to the introduction of the $P^3F$ at session 2. The responses to questions 21, 27, and 32, shown in Figure 5.8(d), suggest a small increase in self-confidence, ability to effectively use subjective information, and orientation towards the overall system goals. Among these questions, the increased agreement from “Not Sure” to “Strongly Agree” on question 21 represents the largest change in agreement, but Subject C responded “Agree” at the fourth session, so the net increase is only 1 point with a downward trend. The subject’s responses to question 29, applying best practices or first principles, had the largest change between sessions 2 and 3 of any question in this section, moving from “Disagree” at session 2 to “Strongly Agree” at sessions 3 and 4. However, the subject also responded “Strongly Agree” at session 1, which was 3 points higher than the section 2 response and essentially nullified the increase at the next session.

Subject C showed strong and consistent agreement with several novice and expert behavioral traits. Using these responses to characterize this subject, he would seem to share several traits with other novice designers that have been observed. In particular, his responses indicate a discrete and concrete functional orientation. He is uncertain about his design decisions until they are embodied in executable code. In contrast to this stated emphasis on the concrete, the subject also indicated a tendency to use various kinds of abstractions and mental models (questions 3, 4, and 6). These seemingly paradoxical responses are characteristics that will be studied in this subject’s verbal protocols.

5.2.3.2 Verbal Protocol

Subject C selected Design Problem 1, the payroll system problem, to work on at the first observation session. The subject began with a review of the problem statement to identify the requirements for the new system. In a sequence of fifteen decisions selecting pieces of information from the problem description, the subject asked one question of the customer: if he could modify the existing database structure or design a new one. After being told he could modify the database as long as none of the existing records were modified, the subject identified two specific modifications he would make. Following these decisions, he began defining a “function” to marshal the information that would comprise a database entry. Fifty-two decisions were coded in this function definition as he described a very specific algorithm for this part of his solution. The definition included details such as how an array of data would be constructed and sorted.
Subject C then asked four questions to gather information about how commissions were entered and calculated. He followed this with more details on his function that handled the commission processing for employees that are eligible for commission payments. His definition continued with a step-by-step explanation of how a paycheck would be computed for each type of employee. This sequence was interrupted with one question about when hourly employee timecards were updated in the system. With that information, he continued to define the specific steps in his function that now seemed to be encompassing all of the system’s functionality. After covering all of the functionality he had initially identified in his review of the problem statement, the subject declared that his design was complete and the next step would be to start the actual implementation.

When the customer asked if he thought he had accounted for all of the requirements, the subject answered in the affirmative and stepped through his function to demonstrate the completeness of his solution. The customer then asked about the calculation of taxes, to which Subject C replied “oh I see that.” He followed this with an explanation of why he had not included this part of the payroll calculation in his design:
You didn't tell me anything about deducting taxes at source. You just told me the deductions and service charges may be deducted and payroll information must be sent to whoever the authorities are. Who whatever they charge for tax something like that.

The subject asked another four questions about how to calculate payroll tax withholding, and then began modifying his function to accomplish these computations. After eight coded decisions describing this algorithm in one part of his function and realizing that he would need to do the same operation in other places, he decided to create a separate function for calculating the taxes to be withheld from an employee’s paycheck. Following the definition of this function, the subject again declared his design to be complete, and stepped through his function to demonstrate this assertion.

Subject C’s behavior during this session was obviously very naive and novice-like. He relied almost entirely on information specifically stated in the problem statement, and took a very linear, implementation-oriented approach to solving the problem. In fact, his solution consisted of two functions: one to compute taxes and the other to do everything else. He did not attempt to describe any user interactions with the system or to gather information about the company’s business processes related to payroll. Overall, his behavior during this observation session were highly novice-like, and the subject did not demonstrate any expert-like behaviors.

At his second observation session, Subject C chose to work on Design Problem 4, the investigation service case management system. After a brief review of the problem statement, the subject began his design by defining a class object to represent a case file. The subject also asserted an upper limit of one hundred items in a single case file. When the customer told him that the actual number of distinct items could number in the tens or hundreds of thousands, the subject seemed surprised that his design would have to cope with collections of things that large. He then began to describe more specifics for the class representing the case file, proposing that the types of things that are known be considered discrete members of the case file class, and that anything added to the case file that did not fit into one of these predefined types should be put into an “unsubscribed” data member.

After considering this decision and its consequences through a series of four information-gathering questions directed to the customer, Subject C offered a different organization for the documents stored in a case file. In this proposal, there would be “four fundamental types of documents that we could come across in any investigation report.” Specifically, these would be text, audio, video, and images. Then each individual document would be categorized using a “specified finite list of descriptors.” This was the seventh time the subject used the term “finite” to try to limit the amount of data his design would need to be able to manipulate. The subject went on to describe how this list would be implemented using a hash table. The customer
cautioned the subject that these details were too specific, and the major concerns were that the system would be reliable, secure, and easy for the company’s investigators to use on a daily basis.

Subject C acknowledged this statement by the customer, but in the next coded decision, he proposed a particular list and table implementation for the keyword descriptors. He was not as specific as the hash table proposal, but only slightly less so. In the next eight decisions, the subject proposed a method for searching for a particular document, but substituting the term “tag” for what he had called “descriptor” and “keyword” in previous proposals. The subject assumed that the only descriptors for any type of document item would be the timestamp, type, and title of the document. The subject seemed to become confused when he was told by the customer that each item would have to have significantly more information associated with it, such as the person who collected the item, when and where it was collected, why the item was added, information about any warrants or other legal actions taken to collect the item, and identification of the individuals in an image or video recording. In response to this new information, the subject proposed associating a second file with each item which would contain all of the descriptive information about the primary document.

Three decisions later, all of which were proposing finer details about this description file, the description suddenly became three files with different sets of tags in each. In another sudden shift of focus, Subject C decided that it would be good to meet with several of the most experienced investigators to find out how they are using the current system of paper-based files. This, he asserted, would be an even better way of designing the system. Just as suddenly, the subject’s attention turned to dumping the contents of a case file for delivery to a client. When the customer asked about the possibility of creating a linked index of a case file contents to make accessing the documents easier for the client, Subject C assumed that this would be a web-based index and began proposing different access control methods. He resisted the idea of using such an index strictly on media such as a portable hard drive or CD/DVD discs. His argument was that it would be much easier for the investigators to create a simple web page rather than have the system generate an index when the case file is to be delivered to the client. This led to a sequence of twenty-one coded decisions specifying various details of the description files so that they could be used in both the search mode and in the case file transmission.

Returning to the problem statement, Subject C’s attention turned to the assignment and management of investigators associated with a particular case. He began his work by assuming that a change involving a particular investigator’s relationship with a specific case would have global consequences. For instance, if an investigator was the lead on one case, and for whatever reason was later removed from the case, that investigator would automatically be removed from all of the cases he or she was associated with. The customer tried to correct this assumption, but Subject C moved on, shifting his focus to recording an investigator’s time spent working on
a particular case for billing purposes. This led to a proposal for a billing record associating an
investigator and a case, functioning as a place for the investigator to record their time. This
proposal was completed in two coded decisions before the subject again shifted his attention
to the process of adding and removing investigators to a case. The subject finalized his design
solution for this part of the system in four decisions that were made based only on the details
given in the problem description.

The subject’s attention, driven by the order of the problem statement, turned to the processes
involved in closing and reopening a case. Relying strictly on the problem statement details,
the subject proposed this part of his solution in three decisions. The final sequence of thirteen
decisions reviewed the proposed system design and described the hardware requirements for his
proposal. The subject also briefly described how the user interfaces would look, but did not
offer to sketch an example.

In contrast to the process orientation exhibited during the first session, Subject C focused
more on a data hierarchy in the development of his design for this problem. The subject’s
behavior during this session continued to be strongly novice-like, particularly in his reliance
on the information that was explicitly given in the problem statement. The subject did have
one episode of somewhat expert-like behavior when he suggested meeting with experienced
investigators to identify how they manage case information in the current system, but this could
also have been a graceful way of avoiding the difficult situation the subject had created through
his proposed method of organizing and searching for case file documents. Subject C’s repeated
desire to limit the number of choices or types of data to a “finite set” is also indicative of his
desire to eliminate uncertainty from his design process. Overall, Subject C’s behavior and design
strategies during the second observation session were still strongly weighted to a novice-like
orientation.

The delivery service system, Design Problem 8, was the choice of Subject C for the third
observation session. Relying only on his first read of the problem prior to selecting it and
starting the actual observation, the subject immediately began identifying data elements for his
proposed solution. This was followed by a high-level description of how his system would work.
During this time, the subject made sixty-eight coded decisions regarding the data organization
and processes for his solution before asking the customer the first question. This question came
at the end of this episode, when the subject stated that the problem was solved and asked the
customer if any more details were desired.

After asking two questions about tips that driver may receive, Subject C began detailing his
understanding of the cash flow from the delivery customer to the driver, then to the company,
and finally to the bank deposit. The subject relied solely on the details given in the problem
statement and his own experience during this episode. The next thirteen coded decisions detailed
what information would be stored in the database to record the cash flow and how his solution
would iterate over the data to create the different reports specified in the requirements. His focus during this time was primarily on the algorithm or processes that would manipulate the data.

Next, Subject C’s attention turned to a database of restaurants that have contracted with the delivery service. He noted that each restaurant’s menu would be stored in this database to facilitate customer order entry and provide a record of what was delivered for each restaurant. When the customer noted that some of their clients change their menus frequently, and that this would require a great deal of overhead for the delivery service staff, the subject became defensive, asserting that it was necessary to have this information stored so that orders could be properly processed in the system. He had significant difficulty accepting this level of uncertainty in the business process.

After very briefly reviewing his proposed design for the customer, Subject C asserted that the system should be ready to use in no more than one week from the approval of the proposal. The subject also said that “I wouldn’t pay any software company more than a week of their pay to do this system.” The customer also asked the subject about the difficulty of adding a means of extracting driver tips from the receipts so that this information could be imported into the company’s existing payroll system. Subject C replied with the proposal to run a script each day to “dump” this information from the database into a suitable file format.

During the third observation session, the subject demonstrated a more even balance between the system data and the associated processes. He was still heavily reliant on the explicit information provided in the problem statement, and seemed to avoid asking questions as a means of minimizing the uncertainty that additional details and requirements might generate. Overall, Subject C was not interested in the business processes related to the proposed system, and directed little of his attention to the users of the system or the environment in which it would operate. Subject C also ended this session after forty-five minutes, stating that he had done all that he could without actually writing code and creating the databases. On the whole, his approach to solving this problem and the solution he described were superficial at best, and the subject did not exhibit any expert-like behaviors or strategies that were expected after the introduction of the $P^3F$ during the previous session.

At the fourth observation session, Subject C selected Design Problem 5, the e-commerce customer management system, as his problem to solve. After a brief review of the problem statement, the subject began describing how his solution will work. It is important to note that he has not yet proposed a solution. Through a sequence of twenty-seven coded decisions, the subject presented his detailed view of how his solution would function and the information that will be manipulated by it. This was done solely on the basis of the problem statement: no questions were asked of the customer. The final coded decision at the end of this sequence states “I guess we have accumulated all parts of the problem into one and have solved it.”
The customer responded with additional details alluded to but not explicitly stated in the problem description. These details described functionality that would automatically suggest items to a registered customer based on past purchases, wish list items, and user-defined preferences. Subject C replied immediately with a proposal for incorporating two additional modules into the system: one to keep track of items that need advertising and another to associate customer preferences and history with the items that need to be advertised. According to the subject, these new modules were all that would be required to enable the generation of customized offerings for registered customers. The subject used a sequence of fifty-one coded decisions to describe very specific details about how this subsystem would work and interact with the main system.

While the subject was very specific about the algorithms he proposed for his solution, when prompted for more details about the overall design of his system to allow it to be compared with other proposals, the subject explained that “this would be my general high level model of the system,” but “I am not really sure how this combination could be done because I have no idea of how this could be done.” He also stated that he was unsure about the details of the data analysis required to match preferences with items that need to be sold.

The customer also asked about why the topic of wish lists had not been covered in the subject’s design. Subject C responded he did not understand the concept, so he chose not to address it in his design. After the customer explained the concept and how other online retailers used it, the subject proposed the creation of another module or service node. Although he had earlier stated that he did not understand the concept, the subject detailed his design for this wish list module over thirty-four decisions, none of which were questions directed at the customer.

The customer again asked Subject C to review his design because the subject seemed to have stopped working. After providing the review, the subject made the following comments about his design:

As to why I would argue that my system is complete is there is no part of the system which is unknown or partially known. I mean assuming that we know how to implement these services which is again just the Java code. If that can be done there is no information as such information or plan that is partially known based on the requirements that I have.

The subject then asserted that this was his complete project proposal and that to develop the system any further would require multiple personnel. As a result, the observation session was stopped after only thirty-five minutes and twenty-one seconds.

During the fourth observation session, Subject C continued to exhibit several key novice characteristics. He tried to avoid any parts of the problem that were not well defined and
explicitly stated in the problem description, on one occasion even ignoring functionality identified in the problem statement that he was not familiar with. He relied very heavily on the details given in the problem statement, and seemed uncomfortable when he had to gather information directly from the customer. The subject did not attempt to identify or define relationships between different parts of the system at anything more than a superficial level, admitting towards the conclusion of the session that he was only presenting a high-level design and that he would have to delegate the design and implementation of the various parts of the system to others.

Although he claimed to be presenting a high-level design, in actuality he tried to be as specific as possible in most of his descriptions. He detailed aspects of his design such as database fields and data types and particular computational sequences. His approach to solving the problem was linear and based on the order of concepts and requirements in the problem statement. His proposed solution was also linear in nature, ignoring the dynamic multi-user nature of the problem. Overall, Subject C did not demonstrate any significant level of expert-like behavior.

Reviewing Subject C’s performance over the four observation sessions, there is no indication that the introduction of the $P^3F$ had any noticeable effect on the subject. He consistently took a linear approach to solving the problem, guided by the topical order of the problem statements. The solutions he designed were also linear in their structure, appearing more like he was designing a simple, single-user, single-task program than a complex software system. Subject C had a strong tendency to provide low-level details that were not fully supported by the information at hand, and seemed to resist gathering information that was not explicitly stated in the problem descriptions. These behaviors were prominent during his work on all four problems.

5.2.3.3 Design Artifact Quality

Subject C’s overall performance was, at best, fair. While he did show improvements on all but one of the evaluation metrics for the solutions produced in sessions 2 and 3, these improvements were not sustained through the last session and none of his evaluations exceeded a “fair” value. As shown in Figure 5.9, his solution to Design Problem 8 produced during the third observation session was his best effort overall. However, the evaluators both noted a lack of attention to the fundamentals of the problem and a superficial, data-oriented approach in his solution to this problem.

Putting his performance into perspective with respect to the other subject who worked on the same problems, Subject C’s evaluations for Design Problem 1 (session 1) were significantly below the average of the other four solutions. The session 2 (problem 4) scores were in the middle between the other two solutions, and the session 3 scores were equal to or higher than the other solution for problem 8. The evaluations for the final session problem were also substantially
lower than the other solution completed.

This subject’s performance was disappointing, since he was a graduate student with professional experience and had taken advanced courses that covered aspects of large system development. A common theme in the evaluators comments on all four of his solutions was that they were too shallow and superficial, and that he did not attempt to identify critical issues within the problems. Because the improvement shown between sessions 2 and 3 did not continue through the last session, these results do not support the conclusion that the $P^3F$ had an observable effect on Subject C’s design performance.

5.2.3.4 Subject C Summary

Subject C maintained a strong novice orientation across all four observations sessions, as demonstrated by his self-assessments, observed behaviors, and quality evaluations. In the self-assessments, the subject consistently agreed with statements indicating a desire for fixed, concrete information, uncertainty about his decision-making unless he could see code execute properly, and a general orientation towards an implementation-based design.

These perceptions were supported by the behaviors he exhibited during all four observation sessions. Subject C relied heavily on the explicit information provided in the problem statements as well as his own personal experience and knowledge. His problem-solving approach was linear and driven by the order of topics in the problem statements. The subject avoided uncertainty at almost any cost, often by simply ignoring ambiguous or poorly structured parts of the problem.

Subject C’s solution evaluations were, at best fair or average, and predominantly poor. The improvements shown between sessions two and three were not significantly sustained through
the fourth session artifacts. The evaluators commented on the shallow and superficial nature of his solutions to all four of the problems, also noting that the subject did not appear to make any attempt to identify critical issues described in the problem statements. Overall, these results do not provide evidence for any changes in performance, behavior, or self-perception that could be attributed to the introduction of the $P^3F$.

5.2.4 Subject D

Subject D was male, a junior Computer Science major who was taking CSC326 (Software Engineering) during his participation in this study. He had one year of professional experience in software development and had not taken any advanced Computer Science elective courses prior to participating in this study. Subject D worked on Design Problem 7 during the first observation session, problem 4 at the second, problem 1 at the third, and problem 6 at the fourth.

5.2.4.1 Self Assessment Questionnaire

Subject D’s aggregate scores over the four observation sessions for the novice-oriented statements in each of the three self-assessment questionnaire sections are shown in Figure 5.10(a). These scores suggest significant differences in the subject’s perceptions of his design skills and strategies between section 1, viewing the design problem, and sections 2 and 3 which relate to navigating the design space and making design decisions, respectively. The trends for the last 2 sections suggest a continued agreement with these characteristics, while the section 1 scores are trending towards greater disagreement with these statements.

Examining the subject’s responses to the section 1 questions, shown in Figure 5.10(b), a decrease in agreement between sessions 2 and 3 occurred on only questions 5 and 7. In both cases, the subject indicated agreement at session 2, then disagreement at sessions 3 and 4, suggesting that the subject was relying less on an implementation-level view of the system under development. The largest inter-session change in this section occurred between sessions 1 and 2 on question 10 which dealt with seeking a working solution rather than determining the importance of system-related issues. At the first session, Subject D indicated agreement with this statement, followed by a “Strongly Disagree” response at the second session. His level of agreement continued to climb with responses of “Disagree” at session 3 and “Not Sure” at session 4.

The subject’s agreement with all five of the questions in section 2 of the questionnaire was somewhat consistent across all four sessions, as shown in Figure 5.10(c). Subject D strongly agreed with question 16 at all four sessions and with question 12 at all but the first session. Question 17 shows a decrease in agreement (from “Strongly Agree” to “Agree”) between sessions
2 and 3 when the $P^3F$ was introduced, but the continued agreement with this statement, combined with the overall level of agreement with the statements in this section, suggest that this decrease was not a consequence of the $P^3F$ intervention.

The slight reduction in reliance on implementation-level views continued in the responses to the questions in section 3 of the instrument. The largest change between sessions occurred with question 25 between sessions 1 and 2, as shown in Figure 5.10(d). In the first session, the subject indicated strong agreement with the execution and evaluation of code as a means to achieve certainty in decision-making. At the second session, he indicated he was not sure about this behavior, but agreed with it again at sessions 3 and 4.

Subject D expert-oriented aggregate scores, shown in Figure 5.11(a), suggest significant increases in agreement with statements in all three sections of the questionnaire. At the first session, the scores for all three sections imply overall disagreement with the expert-oriented statements, followed by sharp increases in the section 1 and 2 aggregates and a further decrease
in overall agreement in section 3. The scores for both sections 1 and 2 continue to increase over the course of the study, but the increase is not as pronounced after the jump from session 1 to session 2, suggesting that this increased agreement is not a result of the \( P^3F \). The section 3 aggregate score, however, showed the sharpest increase between sessions 2 and 3, countering the session 1 to 2 decrease, and continuing to increase at a smaller rate between sessions 3 and 4. Interestingly, each section had one question where the subject indicated consistently strong agreement throughout the study.

In section 1, this was question 1, as shown in Figure 5.11(b). Subject D initially strongly disagreed with question 3, and disagreed with the other 3 expert-oriented statements in this section. Question 6, involving the use of multi-level models, had the largest increase between sessions 2 and 3, where the subject’s self-assessment changed from “Disagree” to “Agree.” This is the only question in this section that might be attributable to the introduction of the \( P^3F \).

The slower rate of change evident in the aggregate scores for section 2 are reflected in the overall higher incidence of agreement with the questions in this section, as illustrated in Figure 5.11(c). In this section, the subject’s responses to question 15 show a more significant change between sessions 2 and 3, coinciding with the introduction of the \( P^3F \). At the first session, Subject D disagreed with the statement that being able to consider multiple issues at once helped him be more productive. At the second session, he indicated that he was not sure about this statement, and at the third and fourth sessions he responded with “Strongly agree.” None of the other questions in this section of the questionnaire had the same degree of increased agreement between sessions 2 and 3, although three others, questions 13, 18, and 19 showed a 2 point increase between the first and second sessions.

As noted above, section 3 showed the sharpest increase between the second and third sessions. This steep slope was driven by Subject D’s responses to three questions, all of which he strongly disagreed with at session 2. Responses to questions 21 and 29, shown in Figure 5.11(d), reflecting the subject’s decision-making confidence and use of first principles and best practices, changed to “Not Sure” at third session and continued to increase to “Agree” at session 4. Subject D also indicated strong disagreement with question 27, relating to his use of subjective information, at the first and second sessions, changing to “Strongly Agree” at the third session and “Agree” at the fourth. As the \( P^3F \) specifically encourages principle-based decision-making and the incorporation of all available information, the responses to these question may indicate an effect of introducing the framework.

Subject D did show consistent agreement with several novice and expert design strategies. In the area of navigating the design space, his responses suggest that he is narrowly focused (question 14), desires detailed, concrete information about system requirements (question 16), and is guided by his own knowledge and experience (question 17), common characteristics of novice designers. Subject D’s assessments of his design decision making processes reinforce these
characteristics, as he indicated steady agreement with statements 22, 23, and 26.

Subject D’s responses to questions oriented towards expert strategies indicated consistent agreement with only three characteristics. His strong agreement with question 1 suggests that he tries to maintain a systemic view of the design problem that incorporates non-software factors. The strong agreement with question 32 further reinforces this orientation. However, overall organizational goals and purposes and other non-software information are rarely included in requirements documentation, contrasting this self-assessment with his desire for explicit and detailed information about the problem at hand. This subject also indicates that he feels more productive when he is able to maintain a flexible and opportunistic design process (question 11).

Overall, the self-assessment data for Subject D suggest that the introduction of the $P^3\mathcal{F}$ may have had some impact on his design decision-making process. In particular, he indicated increased confidence in his decision-making and understanding of the consequences of his decisions (question 21) as well as recognizing the value of fundamental design principles and best practices as a means of supporting his decisions. Subject D’s agreement with several novice-oriented statements also corroborates prior research results.
5.2.4.2 Verbal Protocol

At the first observation session, Subject D chose to work on Design Problem 7, the restaurant information systems problem. The subject began the session by reviewing the problem statement and categorizing entities and concepts he found there. Categories included inventory items, business processes, and information transfer processes. Once he had gathered and categorized all of the obvious information in the problem statement, he began asking questions about the overall business process from the retail customer’s order to its delivery. The subject paid close attention to how the order information gets to the kitchen to be prepared and how the cashier is notified of a completed order, which he identified as a potential bottleneck in the order fulfillment process. Over the course of a eighteen decision sequence, Subject D asked questions and offered alternative methods of transferring this information back and forth to identify the ways that might work best and introduce the least disruption into the current process. The subject noted that this was an important aspect of the system as it relates to the order processing, and having different ideas about what would work best would be helpful. This episode concluded with a decision to meet with the kitchen and cashier staff to get their input about this information transfer.

Deferring the human-oriented details of the cashier to kitchen information interface to a later meeting, the subject began to focus on the hardware interface for whatever subsystem was chosen for the kitchen area. This episode was brief (four decisions) and ended when the subject decided that he needed to turn his focus towards the software side of the system. Subject D began exploring the required functionality of the point-of-sale part of the new system with attention to how this part would interact with the inventory subsystem. This led to a sequence of four information-gathering decisions (questions) about how the inventory is currently managed. The subject’s attention then shifted abruptly to credit card processing at the point-of-sale terminals. Specifically, he was interested in what reports the credit card machine provided and whether or not it had an API that he could access from within his proposed system. This concluded the subject’s high-level information-gathering process, and he said he needed to focus on consolidating the information he had gathered.

However, Subject D’s attention turned to the inventory system, specifically noting the need for a complete list of all menu items and their ingredients and proportions. After asking five questions, he proposed additions to the database structure to keep track of the ingredient inventory. During this sequence of proposal decisions, the subject differentiated between the items that the restaurant orders from its vendors and the items that the customers order from the restaurant. This seemed to be an important realization for the subject, as it established the basic relationship between a menu item ordered by a customer and the ingredients in the restaurant’s inventory that were required to produce the menu item. Subject D used this
information, over a sequence of six coded decisions, to complete his solution proposal.

Subject D exhibited both novice and expert behaviors during this observation session. He did not explicitly state an overall plan for approaching the design problem. However, his initial sequence of information gathering and tentative proposals appeared to help him construct a high-level mental model of the business operations that seemed to guide much of his later work. The proposals offered during this episode had a trial-and-error feel to them because they did not seem to build from what was learned from the previous proposal evaluation. However, the subject did appear to be holding what was learned in reserve so that he could use it later in the design process.

The subject did shift focus without warning on several occasions to parts of the problem/solution that were not logically connected to what he was working on before the shift. Because of the lack of connective information, it is difficult to classify these shifts as opportunistic. Subject D was eventually able to fit the information gained at each of these discontinuous shifts into his high-level view of the system. The subject’s professional work experience prior to participating in this research study is a likely influence with respect to the expert-like behaviors demonstrated at the first observation session. However, the subject did exhibit novice-like behaviors as well.

Subject D chose to work on Design Problem 4, the investigation service case management system problem, at the second observation session. As he did during the first session, the subject began working on this problem by reviewing the problem statement and classifying the information it contained. This time, however, he explicitly stated that he was focusing on identifying data objects that would be used by the system he would design. After a sequence of seventeen decisions categorizing this information from the problem statement, the subject asked the customer if there was a preferred development methodology, specifically agile versus up-front planning. After being told that this decision was his choice, the subject began asking questions about the legal process in general and the company’s activities within that framework. This sequence of thirteen coded information-gathering decisions provided the subject with sufficient information to allow him to proceed with developing his design solution proposal.

Subject D’s attention returned to the data objects he had identified earlier and he began trying to fit them into his understanding of the company’s investigative and reporting processes. His first focal point within this area was the “lead investigator” as described in the problem statement. The subject devoted eleven questions to gathering information about this person’s responsibilities, their expected interactions with the new system, and how this person is assigned or removed from a particular case. Subject D’s attention then shifted suddenly to the circumstances and process for closing a case. This shift seemed to be driven by the text of the problem statement more than an internal design process because he referenced a specific statement in the text. Five information-gathering decisions were coded in this sequence.
The subject then shifted attention again, this time to the different kinds of cases that the company investigated. At this point, Subject D was not interested in the specifics of the different kinds of cases, only that there were differences and that these differences could be captured as attributes of a case record. Focus shifted again to the process of assigning a default lead investigator to a reopened case, and again this shift appeared to be driven by the problem description rather than the subject’s planned approach to solving the problem. Subject D’s attention shifted once more, this time beginning a sequence of five information-gathering decisions about the “case repository” identified in the problem statement. Citing concerns about how to store the quantities of information described in the problem statement, the subject next started to consolidate what he had learned into a Unified Modeling Language (UML) diagram.

At this time, Subject D identified a conceptual entity he referred to as a “document object” that would be used as a generalized representation of an item stored in the case repository. Over the course of the next twelve coded decisions, the subject decomposed his general document object into a hierarchy of more and more specific document types, beginning with a distinction between text-based and multimedia-type documents. This was further decomposed into reports, statements, and scanned text documents under the “text” type and items such as photographs, audio recordings, and digital video files under the “media” type. Additional levels of decomposition were also performed at this time. The results of this decomposition sequence appeared to trigger concerns in the subject about how these different types of documents could be searched, accessed, and organized within the record of a particular case. Over a sequence of seventeen coded decisions, the subject developed and proposed a multi-level filtering system to facilitate searching and browsing the information associated with a particular investigation.

After developing this high-level view of the information management functionality of the new system, Subject D returned to the problem statement to review the system requirements and reconcile them with his developing design. His focus at this time settled upon the investigators and to addressing the need for some way of classifying their expertise. After identifying a high-level classification structure for investigators, the allotted time expired. The subject summarized his proposal by briefly describing the classification system that he had proposed, noting that more information was needed to develop them into more workable structures.

Subject D’s behavior during this observation session also demonstrated both novice and expert behaviors. As in the first session, the subject was not deterred by incomplete information, but generally took a layered approach to discovering finer details, examples of expert-like behaviors. He did appear to be driven more by the ordering of the problem description than by an internal plan for developing a solution, and he did not make explicit efforts to identify and incorporate relationships between the different parts of his proposed solution. Another example of novice-like behavior was the overwhelming amount of effort and time spent collecting
information compared to developing anything more than a very high-level solution. Overall, the subject’s behavior during this observation session was more novice-oriented than in the first session.

Design Problem 1, the payroll system problem, was the choice of Subject D for the third observation session. As with the first two sessions, the subject began with a review of the problem statement in order to identify important concepts and requirements. At this session, however, his initial emphasis was of differentiating between entities that were internal to the new system and those that were external to it. Within this categorization, he tried to organize entities at common levels of scale, specifically referencing the $P^3F$ principle three times. The subject was also attentive to the relationships between these entities, noting key associations such as the company’s position as a “middleman” in the collection of payroll taxes and union dues.

As he began categorizing different types of employees, Subject D also described his planned approach to the start of his design process: “trying to not dive deep,” “trying to make connections between things,” and “trying to identify interactions between different things and how they fit together.” The subject then began to identify concepts in the problem statement that he considered ambiguous, such as the current process being used by the company. Focusing on this process, Subject D began questioning the customer in a sequence of fourteen information-gathering decisions to identify the most critical pieces of the process. Notable among these questions were ten that dealt specifically with how individuals in the company might interact with the new system compared to the payroll-related tasks they currently perform. This led to a seven-question information-gathering sequence concerning the interactions that occur between the company and their bank to process the direct deposit of paychecks. At this point, the subject began to sketch out a view of the interactions that had been identified as a preliminary foundation for his system design.

Subject D then continued gathering more detailed information about time reporting, payroll deduction calculations, paycheck printing, and disbursement of funds to external entities such as tax agencies and employee unions. The subject then proposed an off-the-shelf software solution (Quicken) as a straightforward method of automating most of the payroll-related tasks that were currently being done by hand. The subject noted several advantages of this solution, including low cost and faster implementation. However, the subject also identified several unknowns about this proposal related to whether or not the software has tax calculation and reporting facilities built in to it and the degree of customization that would be available. The subject postponed a detailed investigation of this alternative until more information about the customer’s needs were gathered and understood.

The subject next directed his attention to the information storage requirements for the system, specifically with respect to long-term storage of payroll data. This was a short episode,
lasting only four decisions. Subject D then reviewed the information added to what he gleaned from the problem statement to identify new issues that needed to be addressed to refine the solution alternatives. He focused on the implied requirement that the system be available to more than one user at a time, with different users performing different tasks within the system. His stated goal was to identify the maximum number of different tasks that might be performed simultaneously. With this information, Subject D reevaluated his earlier alternative solutions, eliminating one due to that solution’s inability to handle multi-user access to system data.

After three questions regarding the company’s data backup procedures, the subject again reviewed the alternatives he had documented. His driving question was “do I have enough information to start implementing a design?” Subject D appeared to realize that he did not know enough about the business in general and the primary processes that were used in the day-to-day operations of the company. A sequence of eight information-gathering decisions led to the identification of two additional categories of data to be recorded and reported: sales and commission data. The subject also asked questions about the future plans for the business, particularly regarding expansion and the effects that could have on the payroll system. The customer’s preference to not have to purchase a new version of the system every year or two, combined with becoming locked into a particular data storage format led to the elimination of off-the-shelf software as an alternative. The customer also identified the possibility of expanding into online sales and the desire to eventually have a completely computerized financial records management system. At this point, the session time expired.

The subject definitely exhibited an influence on his design process as a result of the introduction of the $P^3F$. In addition to specifically referencing the levels of scale principle, Subject D demonstrated an understanding of several other principles and of the framework’s design decision-making process. He often developed his definition of a particular entity in terms of how it was related to other entities that were better understood, an example of using boundaries and strong centers. He also demonstrated an iterative approach to solving the problem, frequently returning to review what he had done in terms of where he was currently at in the development of a solution. On the novice-oriented side, the subject spend considerable time gathering information. However, he did identify many more alternatives during the course of this session than he did during the first two sessions. Overall, Subject D exhibited more expert-like behavior at this session than in the first two, particularly the ability to maintain a big picture view of the system that incorporated a variety of aspects of the operational context of the new system, the use of reflection as a tool to review and refine his understanding, and the ability to define boundaries and relationships between different parts of his proposed solution alternatives.

For the fourth observation session, Subject D chose to work on the nursery information system, Design Problem 6. The subject began this session by reviewing the problem statement to identify “relative information” that would be useful to designing a solution to this problem. The
focus of his first six information-gathering decisions was the company’s business environment, particularly the business location, the types and locations of their customers, and the billing process. From this episode, the subject proposed three objectives for the system: sales, real-time inventory monitoring, credit checking, and billing.

The subject then turned his attention to the processes involved in selling merchandise to wholesale and retail customers. In particular, he focused on the differences between the wholesale and retail operations, defining a boundary within the “sales” category to differentiate the two different processes. The subject also recognized that the sales process was common to both inventory management and billing, and that the credit check process was a component of the wholesale sales process but not involved in retail sales. Subject D continued to refine his understanding of the wholesale process by identifying key information items that needed to be maintained in order to properly complete a sale transaction that would lead to a billing statement.

Noting a basic understanding of the sales process, Subject D turned his attention to the inventory management part of the problem. His initial focus in this area was on the processes used to fulfill a customer’s order, distinguishing the different kinds of employees that were involved in the process. He did not attempt to define them in terms of users of the system until he had taken the customer through the entire fulfillment process. At this time, the subject asked the customer about what kinds of interactions he would like the different people in this process to have with the new system.

The subject returned to his overview of the system to confirm his understanding of the goals and objectives with the customer. With that confirmation, the subject directed his attention to the billing process, also identifying the individuals involved and the steps necessary to generate and send out statements to the wholesale customers. In the course of identifying the billing process, Subject D also gathered information about the credit-checking subprocess. At the conclusion of this episode, Subject D proposed the possibility of making changes to the existing system rather than developing a completely new one. He noted that this would probably be a much less expensive path to take and would present the least disturbance to the company’s ongoing business processes. The subject did note that the real-time inventory monitoring might prevent this from being a viable solution.

Investigating the inventory management part of the problem in more detail was the next task Subject D undertook. This episode included thirty-five coded decisions, and the subject followed a pattern of asking two or three questions to identify specific aspects of this part of the system, followed by the proposal of one to three alternative solutions which were discussed with the customer. As the subject progressed through this episode, the alternatives he generated began to converge towards a single generalized solution. After reaching this acceptable proposal, the subject turned his attention towards the hardware environment in which the new system
would operate, posing four questions to build a high-level view.

Subject D again returned to his evolved design and notes, reviewing them with the customer to check for critical omissions or misunderstandings. The subject noted that the process he had followed had allowed him evaluate a variety of possible solutions, and that what he was proposing represented the best of all of these. He also identified one other piece of critical information necessary to complete his design proposal: the company’s sales volume. This information would allow him to choose the appropriate data storage system to provide the performance the customer expected. With this decision, the subject declared his proposed solution as complete as it could be without meeting with various users and directly observing the company’s processes to further refine the proposal.

Subject D exhibited more influence of the $P^3F$ during this final observation session than he did during the third session. He explicitly mentioned three different principles (levels of scale, boundaries, and strong centers) and continued to use an iterative approach to developing a solution that was well situated within the intended operating environment. He was much less “data object” oriented during this session than in the first two, focusing most of his attention on various business processes, the people involved in them, and how these processes interacted and depended upon each other. His sequences of information-gathering decisions tended to be more tightly focused and incorporated more alternative generation than he demonstrated in the third session.

The subject also identified a plan for approaching the problem early in the session, and continued to refer to it and to refine it throughout the session, driven by what he learned from information gathering and from the customer’s feedback on his proposals. Subject D also demonstrated the ability to effectively frame different aspects of the design problem by characterizing the boundaries between these elements. Overall, the subject’s performance during this session exhibited strong expert-like characteristics.

During the first two observation sessions, Subject D exhibited several expert-like behaviors which were probably an effect of his professional software development experience. When the $P^3F$ was introduced to the subject, he immediately seemed to “get it” more than any of the other subject did. At the start of the third session when the framework was reviewed and the subject was given the opportunity to ask questions about the $P^3F$, Subject D made several insightful comments relating the framework and his experiences. His performance over the last two sessions corroborates this apparent interest, since he explicitly incorporated some of the principles and the conceptual process into his design process.

As noted in the discussion above, Subject D’s expert and novice type behaviors were somewhat constant between the first two sessions. At the third session, his expert-like characteristics increased significantly, accompanied by fewer examples of novice behavior. The fourth session represented an additional increase in expert-like design behavior with very little novice charac-
teristics noted. The evidence discovered in the analysis of these verbal protocols suggests that the introduction of the $\mathcal{P}^3F$ did induce an increase in expert-like behavior and a decrease in novice-like behavior.

5.2.4.3 Design Artifact Quality

Subject D showed substantial improvements on all but one of the metric evaluations between the second and third sessions, as shown in Figure 5.12. Four of the metrics for session 3 (problem 1) were “very good” and all eight of the evaluations were better than the average for this problem. However, these gains were not sustained in the solution produced at the fourth session. Furthermore, three of the metrics did not exceed the evaluations made on the first session problem.

![Figure 5.12: Subject D Design quality evaluations](image)

A common theme in the evaluators’ comments on this subject’s artifacts was the evidence of critical thinking. They also noted an overemphasis on hardware aspects of the system on the first two solutions. The shortcomings noted in their comments included ignoring key details critical to understanding how the proposed systems will be used, particularly with respect to the interactions with various users and other existing computer systems.

The evaluator’s comments on Subject D’s third and fourth solutions recognized a more holistic and systems-oriented approach to developing the designs. These comments were made in relation to the subject’s consideration of off-the-shelf software solutions instead of developing a new custom system, evaluating usability vs. speed trade-offs, the use of an agile development
methodology to help ensure customer satisfaction, and network communication concerns.

While Subject D’s performance did not continue increasing through his solution to the fourth problem, the evaluators did identify aspects of his solutions that indicated, to them, a higher level of thinking than was evident in his first solutions. One of the key elements of the $P^3F$ is the emphasis on viewing the system under development as part of a larger whole, and the evaluators’ comments provide support for the conclusion that the framework may have had a positive effect on this subject’s design thinking.

5.2.4.4 Subject D Summary

The overall view of the results for Subject D suggest that the introduction of the $P^3F$ did have a positive effect on the subject, increasing his expert-like behaviors and decreasing his novice attributes. Compared to the other four subjects in this study, Subject D was working at a more expert-like level prior to the start of the study. This was reflected in his self-assessment responses where he consistently asserted trying to maintain a system-level view of the problem and solution in context and the ability to apply a flexible and opportunistic approach to solving complex problems.

These perceptions were substantiated by his observed behaviors. Subject D did demonstrate several expert-like behaviors during the first two observation sessions, in particular the ability to accept and manage uncertain and ambiguous information through the use of abstractions and limited mental models. During the third and fourth sessions, the subject made several references to $P^3F$ principles and demonstrated his ability to use them effectively as design tools.

In their evaluation comments on Subject D’s solution artifacts, the evaluators noted evidence of critical thinking processes on all four artifacts. The third and fourth solutions also exhibited a holistic, system-level view, the consideration of novel solution alternatives, and the evaluation of trade-offs as indicative of even higher level thinking.

Subject D’s initial high level of performance is likely due to his professional experience. This also set the initial bar higher when looking for changes after the second session. However, the subject significantly improved his expert-like characteristics as measured by all of the data collected at the third and fourth sessions. This analysis strongly suggests that the introduction of the $P^3F$ to this subject was the principal cause of the changes in his behavior, self-perception, and design performance.

5.2.5 Subject E

Subject E was male, a junior Computer Science major who was taking CSC326 (Software Engineering) during his participation in this study. He had no professional experience in software development and had not taken any advanced Computer Science elective courses.
Subject E worked on Design Problems 5, 7, 1, and 3 during observation sessions 1, 2, 3, and 4, respectively.

### 5.2.5.1 Self Assessment Questionnaire

Subject E’s aggregate self-assessment scores for the novice-oriented questions, shown in Figure 5.13(a), do not support the conclusion that the introduction of the $P^3F$ had any significant effect in reducing the subject’s novice-like attitudes. The rising agreement scores for sections 1 and 2 suggest that the subject’s self perceptions are actually trending more towards the novice side of the spectrum rather than towards expert-like behaviors and strategies. The aggregate scores for section 3 do show a continual decline in agreement after the $P^3F$ was introduced at the second session, but the score at the fourth session is still higher than at the first session, a net increase in agreement with novice strategies.

Among the novice-oriented questions in the first section of the questionnaire, graphed in Figure 5.13(b), Subject E expressed disagreement with only statement 2, which concerned maintaining a narrow view of the system. He consistently indicated agreement with the other four questions in this section, suggesting a novice-like orientation towards implementation instead of abstract modeling and a desire to restructure problems to fit known solutions. These tendencies fit with known characteristics of novice designers.

Subject E’s responses to questions in the second section of the questionnaire also trended towards agreement with novice characteristics, as shown in Figure 5.13(c). He asserts disagreement with the statement that he is productive when he concentrates on one issue or element at a time (question 14), which correlates with his disagreement with question 2 in the first section. The subject’s responses to questions 16 and 20 suggest his need for well-defined and detailed requirements information and a tendency to use a trial-and-error based approach to solving design problems. These behaviors have associated with novice designers in earlier studies. Subject E’s initial response to question 17 was “Strongly disagree,” but was indicated as “Agree” at the other three observation sessions. This question relates to controlling the design process based on one’s own knowledge and experience, and the subject’s agreement with this behavior correlates with known characteristics of novices[21, 148].

Figure 5.13(d) presents Subject E’s responses to the novice-oriented questions in section 3 of the self-assessment questionnaire. As with the other novice questions, this subject’s responses do not indicate any observable effects of the $P^3F$. On the contrary, the data imply that his self-perceptions changed very little over the course of the study. The subject’s consistent disagreement with question 28 suggests that he is not locked into the results of his decisions. His even agreement with questions 22 and 25, combined with his overall agreement with questions 23 and 26, are evidence of a reliance on his own internalized knowledge and experience and a need
to see working code to have confidence in his solution. These are documented characteristics of novice design behavior[10, 154, 227].

![Graph](image)

**Figure 5.13: Subject E novice-oriented self-assessment scores**

The aggregate scores for Subject E’s responses to the expert-oriented questions on the self-assessment questionnaire, shown in Figure 5.14(a), also present an inconclusive picture of the effects of introducing the $P^3 F$ to this subject. The scores for the section 1 and 2 questions do indicate a slight increase in agreement with expert strategies between the third and fourth sessions, but this follows flat and decreasing slopes between the second and third sessions. These aggregate scores do not support the conclusion that the $P^3 F$ helped the subject behave more like an expert designer. The aggregate score for section three does increase more sharply between sessions 2 and 3, but declines again at session 4 to a point at the same rate of change as between the first and second session scores.

Subject E’s responses to the section 1 questions, presented in Figure 5.14(b) depict overall uncertainty about his perceptions of expert-like characteristics. Questions 4 and 9 are exceptions...
to examine more closely. Both questions deal with the use of abstractions in viewing a design problem. Question 4 addresses this concept explicitly, and Subject E indicates consistent agreement with this statement. The use of mental models is the topic of question 9, and the subject’s self-assessment does change from “Not Sure” to “Agree” across the interval when the $P^3F$ was introduced. However, the level of agreement does not continue to increase. The flat slope between the third and fourth sessions, combined with the uniform agreement with question 4 across all four sessions diminishes the ability to infer that the question 9 change is a result of introducing the $P^3F$.

The subject’s responses to the expert-oriented questions in section 2 are also mixed, with one consistent disagreement, one consistent agreement, one mostly consistent agreement, and uncertainty on the remaining two questions. Subject E’s responses to these questions are shown in Figure 5.14(c). His unwavering disagreement with question 13 indicates that he is not tolerant of uncertainty or ambiguity as are most expert designers. The subject’s consistent agreement with question 15 asserts that he can handle multiple issues or elements within a problem at the same time. After expressing uncertainty at the first session, Subject E persistently states his agreement with the statement that he maintains a flexible and opportunistic design process (question 11).

In his responses to the questions in section 3 of the questionnaire, shown in Figure 5.14(d), Subject E asserts expert-like decision-making strategies, consistently agreeing with four of the six questions. The responses to questions 21, 24, 30, and 32 depict a designer who confidently makes decisions, is open to change as new information becomes available, and is creatively oriented towards large-scale goals driving the need for a system. He also acknowledges a change in his decision-making process on question 29 between observation sessions 2 and 3, moving from a “Not Sure” and “Disagree” position at the first two sessions to agree with the statement at the third and fourth sessions. As this question deals with the use of first principles and best practices which are an integral part of the $P^3F$, it is possible that the introduction of the framework could have influenced this change.

Overall, this subject’s responses to the self-assessment questionnaire do not provide strong evidence for a change induced by the introduction of the $P^3F$. Additionally, several groups of responses seem contradictory. For example, he indicates that he uses abstractions and metaphors to simplify his view of a problem, (question 4), but also desires as much concrete and detailed information as possible and a preference for designing with executable code (questions 5, 7, 10, 16, and 26). Subject E also associates himself with the use of a pre-defined process for problem-solving based on a trial-and-error approach (question 20), but also makes the claim of being flexible and opportunistic (question 11). Finally, he characterizes his decision-making process as deriving solutions that follow directly from the available information (question 22) while also indicating that his decisions are often creative and do not logically follow from the
information at hand (question 30). These conflicting self-assessments raise concerns about the validity of this subject’s responses to the questionnaire, but the verbal protocols provide a record of the subject’s actual behaviors that will help to resolve these concerns.

5.2.5.2 Verbal Protocol

At the first observation session, Subject E chose to work on Design Problem 5, an online e-commerce customer management system. He began the session by rewriting the requirements and goals listed in the problem statement into paragraph form that he felt he could understand better than the bullet list in the problem statement. He also summarized his understanding of the high-level goals for the system:

*I think the overall thing they want done is so that they can keep track of customers better and to help customers with their experiences as well as be able to send catalogs.*

The subject’s attention then fell on to the customer users, as opposed to the company’s administrative users, of the system, identifying two kinds of users and the tasks or use cases...
available to both. Next, he focused in on the details of these use cases, seeking to identify low-level details about their interactions with the online system. His goal for this episode seemed to be differentiating between the two types of customers and how one type (the guest user) could transition into the other (preferred user). This, in turn, led to a continued information-gathering sequence about the specific characteristics of the preferred user type.

At this point, Subject E began developing a UML class diagram to represent his design. He emphasized that he was not going to try to document low-level attributes and operations at this time and focus his attention on gaining a high-level understanding. As he developed this diagram, he started expressing confusion about relationships between parts of the system he felt were important, such as the users, various forms, inventory, and catalogs. He eventually formed connections between the system requirements and his personal experiences with retail websites, and this seemed to satisfy his immediate information needs.

Continuing to develop the UML diagram, Subject E focused on users and the use cases he had documented earlier in the session. During this time, the subject proposed many different details concerning the system-customer interaction. These proposals did not directly extend from the preceding statements and decisions, but seemed to just spring into his awareness. This information-gathering, documenting, and proposing episode lasted 24 minutes and 38 seconds, nearly half of the entire observation session. Although he had stated that he would not try to identify details in his diagram, the integration of the use cases into the UML diagram resulted in the inclusion of these details as operations, attributes, and connecting classes. The session concluded with an explanation, from his perspective, of the relationship between users (customers) and purchases.

While Subject E took a somewhat organized approach to solving the design problem, his focus was almost entirely on one aspect of the system: the users. He did not seek to understand the underlying business value the new system would represent to the company, and did not address many significant aspects of the system design such as securing the website, protecting customers’ privacy and personal information, or the roles of backend users (employees that update the website and manipulate the customer information gathered from it). The subject had a tendency to dig deep for details despite expressing a desire not to do so. He also expressed concern about what he perceived as ambiguity in the problem statement. He also had difficulty visioning and elaborating relationships between different aspects of the system and the desire to make the problem fit within his realm of experience. Overall, his approach to solving the problem was novice-like.

Subject E chose to work on Design Problem 7, the restaurant information systems problem at the second observation session. He began the session by reviewing the problem statement to identify the high-level requirements and goals for the system. He then focused on the point-of-sale process from the business perspective, trying to develop a finite state machine model. As
this progressed, he seemed confused about the relationship between the sale transaction and
the effect of that sale on the inventory. However, he did not ask any questions to clarify this
relationship. Instead, he shifted his focus to the inventory management part of the problem.
This led to another layer of confusion, since the restaurant does not inventory the completed
menu items, only the raw materials for creating those items.

Even after the customer explained the relationship between quantities of meat, vegetables,
and other stock items and the quantities of these materials in various menu items, the subject
still seemed confused, but chose to encapsulate these relationships in something he called
“calculations.” Subject E then shifted his attention to the forecasting functionality described in
the problem statement. This aspect of the design was again glossed over without developing any
significant detail under the heading of reporting. This was an area that the subject seemed to
take a great deal of interest in, trying to develop a variety of different kinds of reports to describe
different areas of the business. One of these proposed reports would detail order processing
time, and led to a proposal for the point-of-sale processing. Although this aspect of the system
had been considered earlier in the session, the subject did not make reference to any of the
information gathered at that time, relying on his current understanding of the order processing
time report.

Abruptly changing focus again, Subject E began working on integrating the various aspects
of the system he had explored into a single model of the system. Within this proposal episode,
the subject returned again to the point-of-sale part of the system, and again did not reference
what had been previously discussed. This time he began the information-gathering process
anew, and continued digging deeper into the details.

Subject E’s approach to this design problem was rather haphazard and superficial. Aspects
that he did not understand were simply ignored. Furthermore, there were many important areas
that were not touched upon, such as different kinds of users, data storage and organization, and
system security. The subject had significant difficulty identifying relationships between different
entities within the problem and within his proposed solution. Overall, Subject E’s behavior and
approach to this problem was very typical of a novice, based on the characteristics identified in
previous studies.

At the third observation session, Subject E chose to work on the payroll system, Design
Problem 1. From the start, the subject expressed confusion about the problem: “there’s like
payroll system the which seems kind of hazy” and “I don’t know what to do with this.” He
mentioned employees, but did not explore that concept further until later in the session, other
than noting that the payroll system depends upon them. Another point of confusion was the
different ways that employees are to be paid, and he spent nearly 25 minutes stating and
restating his understanding of these payment methods and associated processes before finally
arriving at a correct statement. His approach during this episode was obviously trial-and-error,
as he did very little reflection on what he was told was incorrect and just made up new proposals “on the fly.”

Next, he said he was turning his attention towards defining the relationships between the different system elements he had identified, but focused on proposing how to identify different employee types, how their net pay is to be calculated, and how they were to be paid. He did not understand the need to deduct taxes and other amounts, and had a very difficult time with the difference between pre-tax and after-tax deductions. Subject E never clearly defined any significant relationships within the system, finishing with a superficial list of system components with very little detail.

In many respects, Subject E seemed to regress from his already novice-like behavior at the second observation session. His treatment of the problem and his solution were very superficial. It would be hard to say that he was trying to fit the problem into his own experience because it was clear that he did not have even a basic understanding of what a payroll system was supposed to do. The subject avoided uncertainty by simply ignoring it and moving on to another area of the problem; this seemed to be the driving force behind his development plan.

Design Problem 3, the knowledge management system, was the problem Subject E worked on during his fourth observation session. He began the session by “trying to find one big thing to start with and then break it up.” He then began asking questions to gather information about how the business worked and what information is recorded about the consultant employees and their work assignments. However, this high-level information seemed to overwhelm him: “I don’t even know like what bit to start with.” As a means of coping with the information overload, he began to organize the information graphically, in particular trying to capture the basic business processes currently in use. This helped him focus on a key aspect of the problem, that of gathering, storing, and retrieving information about the consultants’ work assignments and other experience that could characterize their work knowledge and be used to select the best consultants for particular assignments.

As a means of delivering this information, Subject E proposed developing a filtered search engine rather than a complex analytical system. He argued that such a system would allow management greater control over various selection criteria, although it would require more personal interaction with the system. This led to further information-gathering about the kinds of experience data to be collected and stored. He made several design proposals regarding ways to collect this information, refining his design through feedback from the customer. This was the extent of his efforts towards a solution, noting that most of what he expected to do would be modifications to the current employee record keeping system and he could not predict what that would entail without looking at that system.

Overall, this subject’s approach to this problem was mixed. From the expert-like perspective, he did pay attention to the big picture, and he used abstract models to help him structure and
understand aspects of the problem and solution. On the other hand, Subject E continued to exhibit many novice-like behaviors. In particular, his approach to designing a solution seemed to be driven primarily by what he did and did not know. He did not try to gain anything more than a superficial understanding of the problem or the existing systems. He also continued to have significant difficulty defining and understanding how different parts of the problem and his solution were related to each other.

While Subject E did show some minor movement towards expert-like behavior during the final observation session, his observed performance seemed very novice-like. The most prominent novice characteristics this subject demonstrated was his superficial approach to the problems. Part of this may be a result of his not being able to handle uncertainty and incomplete information very well. He was also persistent in applying a trial-and-error methodology other than in one episode during the fourth session. Overall, this subject demonstrated very little change across the four observations.

5.2.5.3 Design Artifact Quality

Overall, Subject E’s design quality evaluations, shown in Figure 5.15, were close to the averages, with his solution for Design Problem 3 (session 4) trending slightly above average and those for problem 7 (session 2) trending slightly below. Additionally, all of the evaluations for the session 3 artifacts were an increase from session 2, and these increases were maintained or exceeded by the session 4 evaluations on six of the eight metrics. These results could suggest that the $P^3F$ might have been a factor in the improvement this subject’s design performance.

![Figure 5.15: Subject E Design quality evaluations](image)
Considering the subject’s proposed solution to problem 7 at session 2, the evaluations were consistently 2 to 3 points below those of the other 2 subjects that worked on this problem. The evaluators comments suggest they felt the subject did not understand the problem or how to approach it, with one evaluator noting that the “designer doesn’t know what he/she wants to do” and the other stating that the solution was “incomplete and superficial treatment, representing a basic misunderstanding of the problem.” While they were not as harsh on the first solution, the evaluators’ repeatedly remarked on the lack of details and the apparent lack of understanding of the importance of some of these details.

These comments contrast sharply with those provided with the evaluations of the third and fourth solutions. On these artifacts the evaluators noted a good understanding of the data relationships, workflow sequencing, and a “big picture view of the system in its operational environment.” Both also remarked on the evidence of reflective critical thinking and the beginnings of implementation plans. These comments reflect the recognition of a more holistic view of designing and implementing the proposed solutions. Since cultivating the ability to think holistically is a cornerstone of the $P^3F$, these evaluator comments, in conjunction with the performance improvements evidenced by the evaluation scores, suggest that the framework may be an influencing agent in the subject’s design processes.

### 5.2.5.4 Subject E Summary

Viewed as a whole, Subject E’s self-assessments, observed behaviors, and artifact quality evaluations present a mixed and often paradoxical picture of this subject. Several of his self-assessment responses were at odds with each other. For example, he agreed with statements expressing a need for concrete and detailed information, while also agreeing with statements indicating that he can easily handle uncertainty and ambiguity. He also responded affirmatively to preferring to use a deductive approach to problem solving, but asserted that he is most productive when he is creative and his decisions do not follow logically from the information at hand.

During all four of the observation sessions, his behavior demonstrated a shallow, narrow focus and a strong desire to avoid uncertainty and ambiguity, often by ignoring parts of a problem that he felt were not sufficiently well-defined for him to work on. The evaluations of the solution artifacts from this subject’s first two sessions confirmed these behaviors, with evaluator comments suggesting that the designer seemed confused and did not understand the problem he was trying to solve. Subject E’s solutions for the third and fourth sessions, however, were considered by the evaluators to be well thought-out solutions, demonstrating a good understanding of the problem with a holistic perspective.

The verbal protocol analysis of these two sessions presents a very different view. The third
session was very much like the first and second, with Subject E trying to fit the problem into a solution he was familiar with and appearing to develop a very superficial solution. In the fourth session, the subject was observed to create and maintain a big-picture view of the problem and solution, but the solution he described verbally still appeared to be very shallow. Comparing the actual solution artifacts to the transcript of the session, it was clear that the subject was devoting much more communication effort to his handwritten artifacts than he was to the verbalization of his thoughts. Overall, this suggests that the introduction of the $P^3_F$ may have helped Subject E behave more like an expert in the third and fourth sessions. However, it is difficult to state this with authority because the changes noted by the evaluators were not clearly evident to the same extent in the subject’s self-assessments or in his observed behaviors.

5.3 Summary of Results

This section presents a summary of the results obtained from this research study. Because there were only five participants in the study, attempting to analyze the data to produce generalizable results would not be valid. Therefore, the data collected for each subject was analyzed independently to develop a clear view of each subject’s self-perceptions, behaviors, and performance over the course of the study. The results of these analyses reveal that the introduction of the $P^3_F$ had a clear and positive effect on one subject, may have induced some limited changes in two others, and did not appear to have any effect on the other two subjects. Before discussing these results, the analysis processes are briefly reviewed.

This study produced three primary sets of data: the subjects’ self-assessments of their design skills and strategies; verbal protocols recording their think-aloud work on four different software system design problems; and the artifacts representing their proposed solutions to these problems. The self-assessment data were analyzed directly to identify common traits of each subject and those characteristics that changed over the duration of the study. The verbal protocols were first coded by independent assistants to produce the data for analysis. The analysis of the protocols was a five phase process, described in Section 5.1.2, that involved reconciling the first level codings, identification and labeling of decision sequences, associating non-contiguous sequences, development and refinement of a second-level coding scheme, and a final analysis of the labeled sequences to identify patterns of behavior in each coded protocol. The solution artifacts were evaluated by two experienced software systems professionals to measure each artifact’s quality in terms of eight metrics making up a Design Quality Rubric described in Section 5.1.3.

These different types of analysis provide three different perspectives on the subjects’ responses to the introduction of the $P^3_F$. Triangulating research methodologies and their associated data collection and analysis techniques is especially important for early-stage research such as this study, since it helps to reduce risks and threats to the validity of the research as well as
enhancing the rigor of the work [211]. Of particular interest to the current work, these three distinct sets of data reduce the risk of misunderstood certainty by providing different views that together can provide stronger corroboration than any one of them alone. This corroboration between data sets was a factor in the analysis of two of the subjects in this study, and the lack of clear confirmation between different data leaves those results in a less than certain state.

The results for Subject D provide the most convincing evidence that the $P^3F$ can help novice software designers adopt and apply expert-like design strategies. After he was introduced to the framework, this subject’s perceptions of his own design strategies reflected an increased awareness and ability related to several important expert behaviors. Among these were the ability to consider multiple issues at one time and to use that information to construct multi-level and multi-perspective models of design alternatives. He also asserted an increased reliance on fundamental design principles and domain-centric best practices in his design thinking processes. These expert-like strategies were likely contributors to the increased confidence he reported having in his design decisions.

Subject D’s behavior, as observed through the four study sessions, reinforced these self-perceptions. After the introduction of the $P^3F$, the subject demonstrated an increased awareness of the relationships between different parts of the design problems and his developing solutions, and the ability to define and reinforce these relationships. He also acted more iteratively and reflectively than he did in the first two sessions, frequently revisiting and evaluating previous decisions in light of new information. As part of this iterative and reflective process, he also maintained a clear vision of the context in which his proposed solution would eventually operate.

Finally, the evaluators commented on what they perceived as higher-level thinking processes in his solutions to the problems worked after the $P^3F$ was introduced. As evidence for these perceptions, they commented on his holistic and system-level perspectives on the problems, his consideration of novel solutions (including off-the-shelf software), and his awareness and evaluation of tradeoffs present in the alternatives under consideration. Although the quality scores for the fourth problem were not as high as those for the third one, the evaluators’ comments depicted more expert-like behavior in the fourth artifact compared to the third.

Two subjects showed some changes that might be attributable to their exposure to the $P^3F$, although in both cases the changes were only evident in one of the three data sources. Subject A’s responses to the self-assessment questionnaire remained strongly novice-oriented at all four observation sessions, and the quality of her solutions was consistently low. However, during the third and fourth sessions, after she was introduced to the $P^3F$, she was significantly more organized in her approach to solving the problems. Subject A also tried to establish and refine relationships between different parts of the problems and solutions, something that she did not do at all during the first two sessions. She also made reference to several of the principles presented in the $P^3F$, indicating that she was at least trying to use the framework.
Similarly, Subject E’s self-assessments remained consistently novice-oriented across all four sessions. His observed behavior also appeared to be superficial, highly dependent upon his own experience and the explicit details provided in the problem statements, and averse to uncertainty and ambiguity through the study. The solution artifact evaluators painted a different picture with their scores and comments. They suggested that the subject seemed to be confused during the first two sessions, producing solutions that were very shallow and incomplete. The artifacts from the last two sessions, however, demonstrated a good understanding of the problem, a holistic view of the big picture, and a strong grasp of the relationships between the different parts of his proposed solution.

Subjects B and C showed little to no evidence that they were affected by the introduction of the $P^3F$ in any way. Both of these subjects continued to demonstrate a strong reliance on personal experience and knowledge, a need for specific, detailed, and unambiguous information, and a desire to deliver implementation-level details that they could execute to evaluate. The evaluators also noted that the artifacts produced by both of these subjects were superficial, lacking attention to details stated in the problem description, and making no attempt to identify and respond to critical issues also mentioned in the problem statement.
Chapter 6

Conclusion

This chapter concludes this dissertation with an evaluation of the work through the lens of the research questions presented in Section 1.1. The overarching goal of this research was to develop a learning framework that could help novice software designers adopt and apply expert-like design strategies and behaviors. The specific research questions answered in this dissertation are:

1. What are the key strategies employed by expert designers to solve complex design problems?
2. What are the critical weaknesses in the approaches that novices typically use when attempting to solve difficult design-oriented problems?
3. Which of the identified expert strategies would directly address the critical weaknesses of novice design approaches?
4. How can these expert strategies be incorporated into a novice-oriented knowledge-based learning and application framework?

The first three questions have been answered in the review of the relevant literature presented in Chapter 2. Question 4 is answered in Chapter 3 which describes the foundation, design, and implementation of the Principles, Patterns, and Process Framework. Using the $P^3F$, the following questions are also answered:

5. Are software design students able to understand the basic structure and application of the framework and the decision-making process it incorporates?
6. Are software design students able to demonstrate the expert strategies incorporated into the framework when presented with a complex, ill-structured design problem?
7. Does the framework improve the quality of design artifacts produced by the student?
8. Does the framework improve the student’s perception of his or her design abilities and enable them to be more confident in their design decision-making?

The answers to these questions are discussed in Section 6.1 of this chapter. This is followed by an examination of the limitations of this study in Section 6.3. Section 6.2 summarizes the contributions of this work, and the chapter concludes in Section 6.4 with an overview of future research directions based on the \( P^3F \) in general and the results of this study in particular.

6.1 Conclusions

Question 5 above is answered by some of the responses to the exit survey, shown in Appendix D that was completed at the end of the fourth observation session. Subjects were asked to indicate their agreement or disagreement with the survey statements using the scale 1) Strongly Disagree, 2) Disagree, 3) Not Sure, 4) Agree, 5) Strongly Agree. To encourage the subjects to be honest in their responses and any comments they might make, these surveys were not identified with individual subjects. In retrospect, the insights gained from these responses might have been helpful in understanding how some of the subjects performed during the study if the exit survey had been associated with the individuals. The survey responses relevant to answering question 5 are presented in Table 6.1.

These responses support the conclusion that this group of subjects was, overall, able to understand the basic structure of the \( P^3F \) and to apply it to making software design decisions. One subject did disagree with statement 2, indicating that he or she was not able to relate the principles to software design. The same subject was uncertain about understanding the principles and using the pattern structure and process. Because the exit survey responses were not associated with individual subjects, it is not known which individual made these responses. As noted earlier, if this information was known, it could be helpful in explaining that subject’s performance in the observation sessions. It would seem likely, however, that the uncertainty about the principles could be a factor in her/his inability to apply the framework to software design.

Research questions 6, 7, and 8 are answered by the results of the study presented in Chapter 5. One of the subjects (D) was clearly able to demonstrate expert strategies and behaviors beyond those exhibited in the first two sessions after he was introduced to the \( P^3F \), based on the analysis of the verbal protocols. The verbal protocol analysis also showed that Subject A displayed expert-like behavior to a limited extent in the third and fourth sessions, and made an clear effort to apply some of the \( P^3F \) concepts as she worked on the last two problems. Subject E demonstrated expert-like strategies only during the fourth session. The analysis of the protocols from Subject B and C did not provide clear evidence of expert-like behavior or strategies in any
Table 6.1: Exit survey responses related to understanding the $P^3F$

<table>
<thead>
<tr>
<th>Statement</th>
<th>Responses</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I was able to understand the Principles.</td>
<td>4,5,3,4,4</td>
<td>4.0</td>
</tr>
<tr>
<td>2. I was able to relate the Principles to designing software.</td>
<td>5,4,2,4,5</td>
<td>4.0</td>
</tr>
<tr>
<td>3. The Pattern structure used to document the Principles was logical and clear.</td>
<td>4,4,4,4,4</td>
<td>4.0</td>
</tr>
<tr>
<td>4. The Pattern structure helped me relate the Principles to each other.</td>
<td>4,4,4,3,4</td>
<td>3.8</td>
</tr>
<tr>
<td>5. The Pattern structure enabled me to apply the Principles to design problems.</td>
<td>4,4,3,4,4</td>
<td>3.8</td>
</tr>
<tr>
<td>6. The steps in the Process are logically ordered.</td>
<td>5,4,4,4,5</td>
<td>4.4</td>
</tr>
<tr>
<td>7. The steps in the Process are clearly stated.</td>
<td>5,4,4,4,5</td>
<td>4.4</td>
</tr>
<tr>
<td>8. I was able to use the Process to guide the decisions I made while working on design problems.</td>
<td>4,4,3,4,4</td>
<td>3.8</td>
</tr>
</tbody>
</table>

of their work on the design problems.

The answer to the question, “Are software design students able to demonstrate the expert strategies incorporated into the framework when presented with a complex, ill-structured design problem?” is a conditional yes. After being introduced to the $P^3F$, one subject was clearly able to demonstrate some of the expert strategies incorporated into the framework. Two other subjects also demonstrated expert strategies after their exposure, but to a more limited and less conclusive extent. The results of the verbal protocol analysis of the other two subjects design sessions showed no significant change in their behavior after the introduction of the $P^3F$ compared to their behavior in the first two observations.

Research question 7 asks “Does the framework improve the quality of design artifacts produced by the student?” The answer to this question is positive but less conclusive than the previous question. Based solely on the evaluation scores, several of the subjects did show improvement in the quality metrics of the artifacts produced during the third observation session over those produced for the second session, prior to the introduction of the $P^3F$. However, the scores for these subjects’ second session artifacts were generally much lower than for the first session, and the third session scores were close to the first sessions scores. Also, the scores for the session 4 artifacts tended to show little change or a drop from the session 3 artifact evaluations.

If the evaluators’ comments are taken into consideration, the answer to question 7 is a bit more compelling. Comments regarding the artifacts produced by subjects D and E for sessions 3 and 4 indicate that the evaluators saw evidence of better fundamental designs, even though their evaluation scores did not reflect a higher quality rating. These comments also suggest that the lower scores reflect less complete designs, but that they had higher expectations that,
given more time, these subjects would have produced even better proposals. In this light, the overall results of the design quality evaluations suggest that the introduction of the $\mathcal{P}^3 \mathcal{F}$ can help to improve the quality of the designs produced by novice designers. However, the results also suggest that improvements in quality may lag changes in behavior. I would not consider this to be unusual, since there is almost always a “learning curve” associated with cultivating new skills, and it takes time and practice to become adept and effective.

“Does the framework improve the student’s perception of his or her design abilities and enable them to be more confident in their design decision-making?” is the final research question to be answered by this study. As with question 6, the answer to this question is yes for one of the subjects, and doubtful for the other four. Subject D’s responses to the self-assessment questionnaire indicate a decline in agreement with several common novice strategies and behaviors, and an increase in agreement with several expert characteristics. Most notable among these was that the subject’s responses indicate an increased reliance on fundamental design principles and best practices, and a decreased dependence on personal experience. Subject D’s responses also indicated an increased level of confidence in his design decision-making abilities.

One of the questions that must be asked of any research project is to what degree can the results and conclusions be generalized to the larger population represented by the study sample. In the case of this study, the sample population is too small to be a reliable indicator of general trends in the overall population of novice software designers. Furthermore, only one of the subjects (the graduate student) had the educational background originally defined as a prerequisite for participation in the study, in particular an exposure to developing large, complex software systems as is generally covered in an undergraduate software engineering course.

As a result, no claims of generalizability are made for this research. That does not, however, eliminate the value that this study provides. The results suggest that the framework conclusively helped at least one of the subjects act and perform more like an expert than he did before he was introduced to the $\mathcal{P}^3 \mathcal{F}$, and that this introduction also altered his perceptions of his own design skills and abilities.

6.2 Contributions

For any research project to be considered worthwhile, it must make some contribution to the knowledge base of the field of study. This project makes three significant contributions to the discipline of software design. Two of these contributions are embodied in the response to the research questions and the third is a product of the study design and implementation. This section describes these contributions in the context of the study of software design.

The first contribution is the literature review presented in Chapter 2. The goal of this review of previous studies of expert and novice designers was to identify the most common and
important characteristics of both novice and expert designers. A diligent and multi-disciplinary search of the literature of design did not produce any results that compared novices and experts in design-intensive disciplines and identified the key characteristics of both groups of designers. In light of this, the literature review conducted and presented as part of this work is a novel contribution that examines and summarizes prior research in a form not previously found in the literature. A version of the literature review is under development for peer-reviewed publication and dissemination in more widely accessible format than this dissertation.

The second contribution this work makes is in the design and implementation of the study itself. Prior studies have used various forms of protocol analysis, self-assessments, or quality evaluations, but a study using all three research methodologies could not be identified in a rigorous search of the literature. Designing and conducting this research study was difficult, but the value of having three different kinds of data and analysis proved its worth with the ability to triangulate the effects of the $P^3F^3$ on the research subjects. It is hoped that other researchers will build on the core experimental design of this study and in the long run produce stronger and more reliable research results. Dissemination of this experimental design and the results of this study through a peer-reviewed publication are also under development.

The third major contribution of this work is the Principles, Process, and Patterns Framework itself. This study has shown that this framework can help novice software designers develop expert-like behaviors and attitudes. While these results are not generalizable, they do attest to the value of the work. Furthermore, this framework is unique in the discipline, transcending software development methodologies and extending analytical problem-solving techniques suited primarily for well-formed problems. Elements of the $P^3F^3$ have already been published $[254, 255, 256]$, and a more comprehensive presentation of the framework is under review for presentation at the 2012 American Society for Engineering Education conference.

These three contributions extend the body of knowledge related to software system design and provide a basis for additional related and derivative research projects.

### 6.3 Limitations

An essential aspect of any research endeavor is a critical review of the work to identify the strengths and weaknesses of the project and use these strengths and weaknesses to suggest improvements for later work $[221]$. This section examines the study presented in this dissertation with the goal of identifying the limitations of the research and present changes that would help overcome these limitations in future research.

The most significant limitation of this work was the size of the subject population. With only five participants, the ability to validly generalize the results of the study was effectively nullified. The small number of subjects was due to several factors, but the most critical one
was the timing of the study and subject recruiting. The initial plan for the study intended it to be conducted during a fall semester, with subject recruitment starting the week before the start of classes for the semester. Complications arising from obtaining funds to compensate participants and the necessary Institutional Review Board (IRB) approval for human studies research delayed the ability to begin recruiting subjects until the end of the fall semester. These efforts did not garner any commitments for participation, and additional recruiting was delayed again until the start of the spring semester. Pushing the study into the spring added additional recruiting complications, as most of the target subjects were taking the senior design course as well as preparing for graduation and entering the professional workforce, and could not commit to the time required for participating in the study. As a result, none of the actual participants in the study fit the “ideal” subject for which the study was designed.

Three factors could have helped increase the number of participants and mediated this limitation. First, additional advance planning, particularly with respect to acquiring the funding necessary to complete the research, would have allowed the recruiting efforts to be started earlier. The IRB approval required documentation of funding sources, and recruiting could not begin until that approval was obtained. Second, the recruiting could have been directed more precisely at the “ideal” subjects by gaining recommendations from faculty teaching classes that have software engineering as a prerequisite and making brief recruiting presentations in these classes. Third, since students who have completed the software engineering course are more likely to have part-time work opportunities, increased funding that would allow participants to be compensated at a competitive hourly rate could have helped increase the size of the subject population.

The length of the study was another major limitation of this research, particularly with respect to the time available for the treatment phase when the $P^3F$ was introduced to the subjects. As this study was conducted, subjects were introduced to the $P^3F$ with a forty-five to sixty minute presentation and example work session. Subjects were asked to study the web-based presentation of the framework on their own between the second and third observation sessions, and the framework was reviewed, along with working through additional examples, for the first forty to fifty minutes of the third session. In retrospect, this was not enough supervised time to adequately present and teach the framework and to ensure that the subjects actually understood it. Also, the number of pretest and posttest observations limited the amount of data available to effectively characterize the subjects’ pretest and posttest behaviors and attitudes.

On the surface, the resolution to this limitation is easy: make the study longer, with more observations before and after the treatment phase, and allow for more supervised teaching time and more time for the subjects to internalize the framework. However, this solution is not as simple as it seems. First, it would require additional funds to compensate the subjects. Second, requiring participants to attend more sessions might dissuade them from participating...
at all. This could be overcome through increase compensation. Third, more verbal protocol observations means additional expenses for transcription and coding that must be addressed in the project budget.

Another significant limitation of this study was the distribution of the design problems across the subjects and observation sessions. The rationale for randomly assigning problem sets to the subjects was to limit the opportunity for communication and collaboration outside the observation sessions. Since the original subject population was to be drawn from a more constrained group of students, the likelihood of a subject knowing someone else who was participating in the study would be higher. This could allow a subject who worked on a problem the opportunity to discuss that problem with another participant who had not worked on that problem, giving the second subject a possible advantage over the first. Mixing up the distribution of the problem sets would not eliminate that potential limitation, but would decrease the probability of two subjects who knew each other working on the same problem in close proximity.

A different aspect of the design problem distribution limitation was giving subjects a choice of two problems to work on at each session. As a result, the ability to make comparisons between subjects’ work at a particular observation was significantly limited. This was also an effect of the random distribution of the problem sets. In retrospect, giving the same problem to all subjects at a particular session, with different problems for each observation, would have allowed the subjects to be more reliably compared, even though it increased the opportunity for subjects to help each other by discussing the problems outside the observation sessions. The value provided by the ability to make these comparisons would have outweighed the risks of cross-contamination.

The structure of the self-assessment questionnaire was a less critical limitation of this study. As implemented, the subjects completed the same questionnaire at each session. While they were encouraged to answer the questions without trying to remember previous responses, it is possible that some subjects might do this anyway, especially at the later sessions when they became more familiar with the instrument. The straightforward solution to this would have been to construct different questionnaires by changing the order of the questions and rephrasing questions so that they represented the same self-perceptions but were worded differently to help prevent students from remembering responses given at earlier sessions. This would have complicated the compilation of the questionnaire data slightly, but this could have been overcome through the use of a computerized survey tool.

Two additional limitations of this study involve tools used in the analysis of the collected data. First, the coding scheme given to the independent coders was more complex than it should have been for coders who did not possess the technical vocabulary of the software engineering domain. It would have been better to have a simpler coding scheme that would have allowed
them to identify and code decisions more generically, followed by the application of the coding scheme described in Section 4.3.1.1 as a second-level coding. The analysis would proceed using the same procedure described in Section 5.1.2. The second-level coders would also need to be independent of the study, but should be individuals who are familiar with software design and engineering and the related vocabulary. This would have increased the reliability of the coding and analysis of the verbal protocols. Properly redesigned, these coding schemes could have provided quantitative as well as qualitative data for further analysis.

The other analysis tool limitation involves the Design Quality Rubric defined in Section 4.3.2.1. This rubric was based on one designed to evaluate artifacts that were the products of a semester-long design course. Because subjects had only one hour to work on a design problem, none of them produced artifacts that were as complete as those evaluated against the original rubric. A better choice would have been to develop an original rubric more appropriate to the kinds of solutions that study participants were likely to create. Having completed this study, I now have a clearer vision of what kinds of artifacts to expect from participants in this kind of study, and could develop a set of quality metrics that would be applicable to these artifacts.

A final limitation of this study involves my interactions with the research subjects during the recorded observations. I filled two roles during these sessions: researcher/observer and client/customer. In the role of client/customer, I represented the company or individual soliciting the systems described in the design problems. To avoid injecting bias towards particular solutions, I avoided developing my own solutions to the problems, relying on my years of customer service experience to help me play the role and act like a real customer, enabling the subject to drive the design. However, I did notice that as I played the customer for a particular problem multiple times, I developed a clearer vision of what a customer would expect of the proposed solution. In this study, this might have affected the subject’s solutions and their design processes since I was able to provide more direct responses to their questions during the later sessions. In future replications of this study, I believe that it would be important to refine and document a clear vision for each problem and to have that documentation as a reference when acting as the customer.

6.4 Future Work

As an exploratory study, this research should serve as a foundation for additional studies, both to replicate and corroborate this work and to extend it by focusing on interesting and/or contradictory results. Other areas of inquiry that are derivative of this research include investigating any long-term effects of the $P^3F$ on this study’s subjects, revising the framework in response to feedback from the study participants, developing different versions of the framework to suit different levels of design experience and different design-related disciplines, and investigating
potential non-design applications of the framework.

In the previous discussion of the limitations of this study I identified several areas that I would change before attempting to replicate this work. I feel that a replication study is a vital follow-up to the current work, particularly one that would involve a larger subject population over a longer duration. This would provide the opportunity to corroborate this exploratory work and strengthen the case for bringing the framework into the classroom, either as a dedicated course or as an integral part of one or more design-related courses.

A follow-up study is planned to investigate what, if any, long term effects the $P^3F$ may have had on the participants in this study. The participants in this study were asked at the end of the final session if they would consent to be contacted about participating in a follow-up study, and all five subjects agreed. It should be noted that this is not an informed consent to participate in a future study, but only a consent to be contacted with a request to participate. This follow-up study will be some type of online survey or questionnaire developed to measure any changes in attitudes or self-observed behaviors noted by the subjects.

There are several areas in which the web-based presentation of the $P^3F$ can be improved, and the study participants’ responses shown in Table 6.2 highlight the most important revisions that need to be made. The extension of the framework to incorporate software design and engineering principles, such as those identified in Section 3.2, has been in the planning and development process, and will be one of the first major revisions of the web site after it is migrated to a more permanent home at http://www.p-cubed.info/.

A second major revision in the planning stage is the incorporation of more examples spanning various software-related applications of the principles into the pattern-based presentation. This will be a bit of a challenge to add more content while keeping the presentation as compact as possible. A third major revision of the web site will include refining the existing tutorials and adding additional ones to the site, including video clips, animations, and/or other interactive elements that would provide different presentations to support different types of learners.

I had the opportunity to observe their reactions to and interactions with the $P^3F$ web site, particularly when I was introducing and discussing the framework during sessions 2 and 3. Because four of the five subject in this study did not have the level of academic experience expected when the study was designed, their reactions to and interactions with the $P^3F$ presentation used in this study were especially interesting. As I expected, they had difficulty with some of the terminology distinguishing “programs” from “software systems” and the examples given in the $P^3F$ patterns often did not provide the help they needed to understand the complexities of large systems.

As a result, I began thinking about how to modify the content to make the framework more accessible to users who have not been exposed to large software systems and the terminology of software engineering. While these ideas are still in the embryonic stage, I expect to develop them
Table 6.2: Exit survey responses related to $P^3F$ web site

<table>
<thead>
<tr>
<th>Statement</th>
<th>Responses</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>9. The Principles, Patterns, and Process Framework website is logically organized and easy to navigate.</td>
<td>5,5,4,5,5</td>
<td>4.8</td>
</tr>
<tr>
<td>10. The Principles, Patterns, and Process Framework website helped me learn how to use the Framework to solve design problems.</td>
<td>4,5,4,3,4</td>
<td>4.0</td>
</tr>
<tr>
<td>11. The Principles, Patterns, and Process Framework website should be extended to include other software design and engineering principles.</td>
<td>5,3,3,2,5</td>
<td>4.0</td>
</tr>
<tr>
<td>12. The Principles, Patterns, and Process Framework website should include more examples.</td>
<td>3,3,4,5,5</td>
<td>4.0</td>
</tr>
<tr>
<td>13. The tutorials provided on the website clearly illustrated how to use the Framework to help solve design problems.</td>
<td>4,4,2,3,5</td>
<td>3.6</td>
</tr>
</tbody>
</table>

in the future. In particular, I would like to develop a $P^3F$ version targeted towards high school and first-year computer science and software engineering undergraduate students, drawing on my experiences as a teaching assistant and instructor for early-sequence programming courses. Several colleagues have also expressed interest in this area of application. Dr. John Tector, one of my dissertation committee members remarked that introducing new computer science students to the holistic and reflective thinking processes incorporated into the $P^3F$ may be more important to their ultimate success than waiting until they have been exposed to large, complex systems.

Other design-related disciplines that I believe could benefit from a “customized” version of the $P^3F$ include fields with a strong visual and/or tactile emphasis such as graphic design and visual arts, architecture, and creative writing. Non-design fields such as economics, project management, organizational behavior, and complex systems also intrigue me, and I occasionally have glimpses of how the framework could be used to gain a greater understanding of phenomena in these areas. One other application of the $P^3F$ that I would like to explore is the development of metrics for software systems based on the framework principles. I expect this to be a challenging endeavor because it represents an attempt to quantify the abstract and somewhat subjective values embodied in the principles.
REFERENCES


[73] Cross, N. Design cognition: Results from protocol and other empirical studies of design activity. In Eastman et al. [91], ch. 5, pp. 79–103.


[191] NEWSTETTER, W. C., AND MCCrackEN, W. Novice conceptions of design: Implications for the design of learning environments. In Eastman et al. [91], ch. 4, pp. 63–77.


[205] **Quellmalz, E. R.** Developing reasoning skills. In Baron and Sternberg [25].


[257] Zannier, C., Chiasson, M., and Maurer, F. A model of design decision making based on empirical results of interviews with software designers. Information and Software Technology 49, 6 (June 2007), 637–653.

APPENDICES
Appendix A

Design Problems

Problem Set 1

Problem 1 — ABC, Inc. Payroll System
(Adapted from [175, pp. 293-295])

You have been tasked with designing a batch payroll system for ABC, Inc. The system will consist of a database of the company’s employees, and their associated data (such as time cards). The system must pay all employees the correct amount, on time, by the method specified. Various deductions must be taken from employees’ gross pay, and records of deductions and other payroll-related information must be maintained to support periodic reporting of payroll information to local, state, and federal tax authorities and for the employee’s tax records. Following are notes taken during an interview with the customer.

- Some employees work by the hour. They are paid an hourly rate that is one of the fields in the employee record. They submit daily time cards that record the date and number of hours worked. Hourly workers are paid weekly, with the payroll period starting on Friday morning and ending the following Thursday night. They are paid every Friday afternoon for the week ending the night before. If an hourly worker works more than 40 hours in a weekly payroll period, they are paid 1.5 times their normal rate for those extra hours.

- Some employees are paid a flat salary. They are paid on the last working day of the month. Their monthly salary is one of the fields in their employee record.

- Some salaried employees are also paid a commission based on their sales. They submit sales receipts that record the date and the amount of the sale. Their commission rate is a fields in their employee record. Commissions are paid every other Friday afternoon for the 2 week sales period ending the night before (Thursday).

- Employees can select their method of payment. They may have their paychecks mailed to the postal address of their choice, have the checks held by the paymaster for pickup, or request that paychecks be directly deposited into the bank account of their choice.
Some employees belong to a union. Their employee record has a filed for the weekly dues rate. Their dues must be deducted from their pay. Also, the union may assess service charges against individual union members from time to time. These service charges are submitted by the union on a weekly basis and must be deducted from the appropriate employee’s next pay amount.

The payroll application will run once each working day and pay the appropriate employees on that day. The system will be told what date the employees are to be paid to, so it will generate payment records from the last time the employee was paid up to the specified date.

Problem 2 — Birdie Masters Golf Schools Operations System
(Adapted from [83, pp. 302-303])

The system development project team at Birdie Masters Golf Schools has been working on defining the architecture design for a new system. The major focus of the project is a networked school location operations system, allowing each location to easily record and retrieve all school location data. Another system element is the use of the World Wide Web to enable current and prospective students to view class offerings at any of the Birdie Masters’ locations, schedule lessons and enroll in classes at any location, and maintain a student progress profile — a confidential analysis of the student’s golf skill development.

The project team has also been considering the globalization issues that should be factored into the architecture design. The school’s plan for expansion into the golf-crazed Japanese market is moving ahead. The first Japanese school location is tentatively planned to open about 6 months after the target completion date for the system project. Therefore, it is important that issues related to the international location be addressed now. Following are some notes recorded during discussions with Birdie Masters staff and students, as well as ideas that have developed during the system architecture design work.

Birdie Masters currently operates 78 golf schools in the US and Canada, and employs 236 instructors. Each Birdie Masters school is located at a particular golf course, but instructors may have teaching privileges at other courses in the same local area that are not directly affiliated with Birdie Masters Golf Schools.

There are over 250,000 past and present Birdie Masters students. While Birdie Masters schools are currently located only in the US and Canada, students reside all over the world. Each week, approximately 12,000 students participate in classes or individual lessons at Birdie Masters locations.

Birdie Masters offers both individualized private lessons and “standardized” classes for 3 to 10 students. Standard classes meet either once a week for 5 or 10 weeks (depending upon the class) or daily for 5 consecutive days at locations in “vacation destination” areas. Class topics include: Introduction to Golf, Fairway Shots, Around the Green, Sand Shots, and On the Green. All of the classes except the introductory class are offered at Beginner, Intermediate, and Advanced levels. Other intermediate and advanced classes are offered at individual locations and feature topics related to the unique characteristics of those courses.
Intermediate classes require completion of the associated beginner class or demonstration of equivalent skill level. Intermediate class skills are a prerequisite for advanced courses.

- Currently, students must schedule private lessons and enroll in classes in person at the Birdie Masters location where they will participate. This presents problems for golfers who would like to take a class while on vacation, particularly since these classes often fill up quickly. The ability to schedule lessons and enroll in classes via the web would be a valuable asset to Birdie Masters, and a feature desired by over 80% of the nearly 2000 respondents to a customer satisfaction survey.

- Students receive written evaluations and advice from their instructor for all regular classes completed and are encouraged to keep these evaluations as a personal record of their accomplishments. Over 72% of the survey respondents indicated that being able to maintain these evaluations online rather than on paper would be “very desirable.”

- Market research data collected in Japan suggests that Birdie Masters should expect very high demand for their classes in that market. The current plan involves opening 3 schools initially, with the expectation of adding 3 to 6 locations per year for the first 5 years of Japanese operations. This research also suggests that success in the Japanese market may lead to opportunities in other Far East countries.

- Regional managers and corporate executives would like to be able to monitor schools’ performance through some kind of query and/or reporting functionality. These features would not necessarily need to extend to the level of monitoring individual instructors (which should be done at the instructor’s home school), but would allow managers to view reports for individual schools, aggregate data for groups of schools, etc.

**Problem Set 2**

**Problem 1 — ACME Consulting Knowledge Management System**

(Adapted from [83, p. 21])

You are an analyst for ACME Consulting, Inc., a large software consulting firm with offices around the world. In addition to approximately 7,000 administrative staff employees, the company employs over 80,000 consultants world wide in a variety of different positions. The company wants to build a new knowledge management system that can identify and track the expertise of individual consultants anywhere in the world on the basis of their education and the various consulting projects on which they have worked.

Education is tracked through existing employee records indicating degree(s) earned, continuing education and other college-level courses, recognized certifications and accreditations, corporate training, and other learning opportunities.

Employee records also indicate work assignments and roles and are linked to additional general details about the assigned projects. However, information regarding specific types of tasks performed, technology areas, in-house and client review and comments, etc. that occur in the course of an assignment are not currently recorded. Modifying the existing information systems to record and update this additional information is itself a significant project, but would add valuable information to the knowledge management system.
Assume that this is a new idea that has never before been attempted in ACME or elsewhere. ACME has an international network, but the offices in each country may use somewhat different hardware, operating systems, and software configurations. ACME management wants the system up and running within a year.

Problem 2 — Jones Legal Investigation Service Case Management
(Adapted from [83, p. 540])

The new information system at Jones Legal Investigation Services will be developed using object-oriented technology. The data to be managed by this system will be complex, consisting of large amounts of text, dates, numbers, dates, graphical images, scanned and other electronic documents, email messages, video clips, and audio clips. The primary function of the system will be to establish an investigation when a request comes in from client-attorneys, record the investigative procedures that are conducted and the information that is gathered during an investigation, and produce bills for investigative services.

The case investigation process undergoes several states in the Jones system:

1. An attorney requests an investigation and submits a copy of the existing case file, including interview transcripts, police reports, pertinent client records, etc.

2. A “case record” will be established in the new system as a repository for information collected during the course of the investigation. A lead investigator is assigned to the case at this time.

3. The existing case documentation is added to the case repository, either through secure transfer of electronic files or by scanning paper documents.

4. The lead investigator begins working on the case, recording all information collected with timestamps as well as the investigator’s time spent performing various investigative processes. The case becomes active with the investigator’s first access of the case record.

5. Investigators may be added or removed from a case at any time, including changing the lead investigator, with the constraint that a lead investigator is assigned to it at all times while the case is active.

6. The client-attorney can begin settlement negotiations with the other party or parties involved, or the case can go to trial. Settlement negotiations may result in a settlement, or they may fail, resulting in a trial.

7. The case investigation is closed either when a settlement is reached and approved by the court or by a judicial trial verdict on the case.

8. An investigation may be reopened as a result of an appeal against a trial verdict. By default, the lead investigator at the time when the case was closed will be assigned the lead role on the reopened case, although this assignment can be overridden when the case is reopened.

9. Appeals do not involve settlement negotiations, and will be closed only upon a judicial decision on the appeal.
Problem Set 3

Problem 1 — The Clothing Shack Customer Management System
(Adapted from [245, p. 146])

The Clothing Shack is an online retailer of men’s, women’s, and children’s clothing, and has been in business for 4 years. The company makes a modest profit from its online sales, but has struggled to compete against “heavyweight” online retailers. The Clothing Shack’s marketing director, Mayaka O’Neil, has determined that the company’s marketing information systems need to be improved in order to help boost sales and make the company more profitable.

Ms. O’Neil feels that The Clothing Shack should begin sending out catalogs to its customers, keep better track of its customers’ buying habits, perform targeted marketing, and provide a more personalized shopping experience for its customers. Several months ago, Ms. O’Neil submitted a Systems Service Request to the company’s steering committee detailing the problems with the current system and the features that would be necessary to improve The Clothing Shack’s marketing efforts. The committee unanimously approved the project.

Currently, customers have the option of creating an online account with The Clothing Shack that allows their shipping and contact information to be stored for future use. Customers may also make purchases as a guest, filling in the required information each time they make a purchase. This information is not retained after the transaction is complete, except for the invoice record in the database. However, all purchases made this way are assigned to a single “Guest” account number, and the ability to track information on these customers is limited. Over the past 5 years, only about 40% of the purchases made use a customer-created account, compounding the problem of tracking and analyzing this information.

At the center of the new functionality for customer accounts is the creation of an online equivalent of the preferred shopper cards used by many brick-and-mortar retailers. By offering the opportunity to participate in “private” sales, accumulate points towards additional discounts or free merchandise, The Clothing Shack expects to significantly increase the percentage of customers that create an account on the website. This will result in much richer data on the purchasing habits of their customer, provide updated contact information for periodical catalog mailings, and the opportunity for the company to engage their customers “individually.” In particular, customers who setup an online account will be able to track their own purchases and returns, maintain wish lists for future purchases, and set other preferences for viewing and searching the website.

Problem 2 — Evergreen Nursery Information Systems
(Adapted from [245, p. 188])

Evergreen Nurseries offers a wide range of lawn and garden products to its customers. Evergreen conducts both wholesale and retail operations. Although the company wholesales to nurseries across the country, Evergreen’s founder and president has restricted its retail operations to California, the company’s home state. The company is situated on 150 acres and wholesales its bulbs, perennials, roses, trees, shrubs, and Evergreen Accessory products. Evergreen Accessory products include a variety of fertilizers, plant foods, pesticides, and gardening supplies.

In the past 5 years, the company has seen a phenomenal sales growth. Unfortunately, its information systems have been left behind. Although many of Evergreen Nurseries’ processing
activities have been computerized, these activities require reengineering. You are part of the team hired by Seymour Davis, the company’s president, to renovate its wholesale division, specifically the billing, order processing, and inventory control systems.

From the requirements determination, you are able to document their current wholesale sales process, as follows:

1. An Evergreen Nursery customer places a call to the nursery.
   (a) A sales representative takes the customer’s order.
   (b) The sales representative checks the customer’s credit status in the information system.
   (c) The order is checked against inventory and the customer is informed of the products’ status.
   (d) The customer is informed of any special discounts or other offers that are in effect.
   (e) The customer is given the total amount due for the order.

2. Ordered items are pulled from stock.

3. When the all ordered items have been pulled, the product inventory and customer account are updated.

4. Ordered items are packed and shipped to the customer.

5. The customer’s account is updated with the shipping date for the order.

On the fifteenth of each month (or the first business day thereafter), a billing statement is generated and sent to each customer. The customer has 30 days to remit payment in full; otherwise, a 15% penalty is applied to the customer’s account. Additionally, new orders to the customer will not be accepted until all past due amounts are paid in full.

One of the major changes to the billing system is to split the billing cycle into 2 semi-monthly statements, allowing customers to choose which cycle they are on. The company would also like to be able to track payments to individual invoices instead of the current statement-level tracking. To help make the order process more efficient, Evergreen Nursery would like to have a customer’s credit status show up automatically in the order-entry interface. Currently, the credit check requires a phone call to the accounting department, which creates a delay in processing the order and wastes sales representatives’ time.

Evergreen Nursery would also like to be able to monitor inventory on a real-time basis. This change would impact both inventory control and order processing. While not a frequent problem, there are a significant number of occurrences when a customer has been told that items are in stock, but were actually being used to fulfill another customer’s order that was not complete at the time the second order was entered. This change would require order pickers to enter each item and quantity as they are picked from stock with immediate inventory updating. The order entry system should also be modified place ordered items “on hold” once an order is complete. The final check for in-stock items should occur as part of the order finalization process.
Problem Set 4

Problem 1 — Hoosier Burger Information Systems
(Adapted from [245, p. 34])

As college students, Bob and Thelma often dreamed of starting their own business. While on their way to an economics class one day, they drove by Myrtle’s Family Restaurant and noticed a “for sale” sign in the window. Bob and Thelma quickly made arrangements to purchase the business and Hoosier Burger Restaurant was born. The restaurant is moderately sized, consisting of a kitchen, dining room, counter, storage area, and office. Currently, all paperwork is still done by hand. Thelma and Bob have discussed purchasing a computer system; however, Bob wants to investigate alternatives and hire a consultant to help them.

Perishable food items, such as beef patties, buns, and vegetables are delivered daily to the restaurant. Other items such as napkins, straws, and cups are ordered and delivered as needed. Bob receives deliveries at the back door of the restaurant and then updates a stock log form. The stock log helps Bob track inventory items and is updated when deliveries are received as well as each night after the daily sales have been tallied.

Customers place their orders at the counter and are called when their orders are ready. The orders are written on an order ticket, totaled on the cash register, then passed to the kitchen to be prepared. The cash register is not capable of capturing point-of-sale information; it only maintains running totals of total sales, sales tax collected, and the grand total collected. Once an order has been prepared and delivered to the customer, the order ticket is placed in the completed order box. Bob reviews the order tickets nightly and adjusts the inventory appropriately.

In the past few months, Bob has begun to notice several problems with Hoosier Burger’s current information systems, especially with inventory control, customer ordering, and management reporting systems. Because the inventory control and customer ordering systems are paper-based, errors occur frequently, often impacting delivery orders from suppliers and the taking of customer orders. Bob has often wanted to have electronic access to forecasting information, inventory usage, and basic sales information. This access is impossible because of the paper-based system currently used.

Problem 2 — Waiters on Call Meal Delivery Service
(Adapted from [219, pp. 159-60])

Waiters On Call is a restaurant meal-delivery service started in 2008 by Sue and Tom Bickford. The Bickfords both worked for restaurants while in college and always dreamed of opening their own restaurant. Unfortunately, the initial investment was always out of reach. They did notice, however, that many restaurants offer takeout food, and some restaurants — primarily pizzerias — offer home delivery service. Many people they met liked the option of home delivery, but would really appreciate a wider variety of choices than the pizzerias offered.

Sue and Tom conceived of Waiters On Call as the best of both worlds: a restaurant service without the high initial investment. The Bickfords contracted with a variety of well-known restaurants in town to accept orders from customers and to deliver the complete meals. The contracts specify a wholesale price discount that is charges to Waiters On Call. Customers pay
the restaurant’s regular retail price plus a service charge (and driver tip). The business started modestly, with only three restaurants and one driver working the dinner shift. However, business has expanded rapidly and the Bickfords realized that they need a custom computer system to support their operations that have grown to five drivers delivering from twelve restaurants. They also have a growing list of restaurants who want Waiters On Call to deliver for them. Sue and Tom hired a consultant, Sam Wells, to help them figure out what they need and how to go about getting it up and running.

“What sort of events happen when you are running your business that makes you want to reach for a computer?” asked Sam. “Tell me what usually goes on.”

“Well,” answered Sue, “when a customer calls in wanting to order, I need to record it and get the information to the right restaurant. I need to know which driver to ask to pick up the order, so I need drivers to call in and tell me when they are free. Sometimes customers call back wanting to change their orders, so I need to get my hands on the original order, contact the restaurant to see if the order can be changed and if so, make the change and notify the customer either way.”

“Oh, how do you handle the money?” queried Sam.

Tom jumped in. “The drivers get a copy of the bill directly from the restaurant when they pick up the meal. The bill should agree with our calculations. The drivers collect that amount plus a delivery service charge. When drivers report in at closing, we add up the money they have and compare it with the records we have. After all drivers report in, we create a deposit slip for the bank for the day’s total receipts. At the end of each week, we calculate what we owe each restaurant at the contracted wholesale price and sent them a statement and a check.”

“What other information do you need to get from the system?” continued Sam.

“It would be great to have some information at the end of each week about orders by restaurant and orders by area of town — things like that,” Sue said. “That would help us decide about advertising and contracts with restaurants. Then we need monthly statements for our accountant.”

Sam made some notes and sketched some diagrams as Sue and Tom talked. Than after spending some time thinking about it, he summarized the situation for Waiters On Call. “It sounds to me like you need a system to use whenever these events occur:”

- A customer calls in to place an order, so you need to record an order.
- A driver is finished with a delivery, so you need to record delivery completion.
- A customer calls back to change an order, so you need to update an order.
- A driver reports for work, so you need to sign in the driver.
- A driver submits the day’s receipts, so you need to reconcile driver receipts.

“Then you need the system to produce information at specific points in time — for example, when it is time to:”

- Produce an end-of-day deposit slip.
- Produce end-of-week restaurant statements and payments.
• Produce weekly sales reports.
• Produce monthly financial reports.

“Based on the way you have described your business operations, I am assuming you will need to store information about these types of things, which we call data entities or domain classes.”

“Then I suppose you will need to maintain information about restaurants and drivers. You’ll need to use the system when you add a new restaurant, a restaurant changes its menu or prices, you hire a new driver, or a driver leaves. Am I on the right track?”

Sue and Tom quickly agreed that Sam was talking about the system in a way they could understand. They were confident that they had found the right consultant for the job.
Appendix B

Problem Set Instructions

One of the major goals of this research study is to develop a better understanding of how computer science students approach and solve complex software design problems. A key part of this understanding will result from observing you work on solutions to real-world design problems. This observation will be made through audio-video recordings, and to help capture what you are thinking and doing, you are encouraged to “think aloud” while you work. Thinking out loud means that you should keep talking while you work, speaking whatever thoughts come to mind no matter how unimportant they may seem to you. Also, you should not try to explain or interpret your thoughts to the camera or in-room researcher. Note that the camera will be focused on the table and computer screen to help maintain your anonymity throughout the observation session.

You may ask the in-room researcher questions regarding the problem you are working on, either to clarify material given in the problem statement, or to gather additional information about the problem, the “customer,” or possible solutions. You will also have the following resources available:

- A pad of lined paper
- Pencils and pens
- A computer with access to the following software modeling and development software:
  - Microsoft Visio
  - Microsoft Visual Studio
  - Eclipse IDE with Modeling Tools (Java & C\(C++\))
  - Sun Java Development Kit

This problem set contain two software system design problems. Your task for this session is to develop a solution to one of the problems. You should read through both problems before choosing the one you want to work on, and you may ask questions about either before you start working on the one you choose. You will have 60 minutes to work towards a solution, and we do not expect you to create a complete product. You will document your work and solution using paper and, or computer resources in any combination you prefer. However, you are asked to clearly indicate the part(s) of your work that you want to be considered as your solution.
Solutions may be represented or modeled in any way desired, but should be as clear and understandable as possible. Part of the task is to identify the problem(s) that must be solved. Solution details will vary among the problems, depending upon the type and detail of the information provided. Note that there are no predefined “correct” solutions to these problems. However, the solution you develop will be evaluated for feasibility, appropriateness, and completeness as part of the research into understanding how students like yourself deal with complex software design problems.
Appendix C

Self-assessment Questionnaire

Overview and Instructions

The purpose of this questionnaire is to collect information that captures how you think that you understand, approach, and solve complex software design problems. You are asked to assess your own skills and attitudes about designing software system. There are no right/wrong or good/bad responses to these questions, and we ask that you answer honestly about yourself.

The only identifying information collected as part of this questionnaire is your subject ID number, which will be used for three purposes: 1) to relate your self-assessments with your observed behavior as you solve design problems; 2) to identify any changes that occur over the course of this study with the way you think about software design; and 3) to statistically relate your self-assessments with other subjects that are participating in this study.

The questions are divided into three sets, each relating to a particular aspect of design problem solving. Each section consists of one statement stub followed by a series of different phrases that complete the statement. For each complete statement, you are asked to indicate your level of agreement or disagreement using the scale 1) Strongly Disagree, 2) Disagree, 3) Not sure, 4) Agree, 5) Strongly Agree.

Viewing the Design Problem

The following statements relate to how you view design problems and create models that help you find a working solution. Please indicate your level of agreement or disagreement using the scale 1) Strongly Disagree, 2) Disagree, 3) Not sure, 4) Agree, 5) Strongly Agree.
Each item below completes the following sentence: “To help me create and maintain my view(s) of a large and complex software design problem, I usually...”

<table>
<thead>
<tr>
<th>Statement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>incorporate the application context, the computational domain, and views of all stakeholders.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maintain a focused view of only the part of the system I am working on, guided by my personal perspective and experience.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize abstract experiences and patterns that can be applied to the current problem.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>use metaphors, common names, and other abstract labels to simplify the problem and solution.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>develop solution models using pseudocode, programming languages, or other tools that generate program code.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>construct solution models that incorporate multiple levels of detail.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>develop solution models using pseudocode, programming languages, or other tools that generate program code.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>try to reorganize or restructure the problem so that it resembles problems I have solved before, allowing my to reuse those specific solutions for the new problem.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>create models that can be mentally simulated to evaluate possible solutions.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>am concerned more about getting a working solution running than about which issues are more or less important than others.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Navigating in the Design Problem/Solution Space

The following statements relate to how you manipulate and move around in the “space” containing the design problem and your solution. Please indicate your level of agreement or disagreement with the following statements using the scale 1) Strongly Disagree, 2) Disagree, 3) Not sure, 4) Agree, 5) Strongly Agree.
Each item below completes the following sentence: “As I move from a complex problem to a designed solution, I am more productive when I...”

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>maintain a flexible process that allows me to opportunistically choose what part of the system I am working on at any time.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>define parts of a system in terms of their functionality within their structural location in the system.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>have a high tolerance for uncertainty and ambiguous information.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>concentrate on addressing one issue or element of the problem at a time.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>am able to consider multiple issues of elements simultaneously.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>have access to concrete examples of requirements that incorporate details and information that are as complete as possible.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>control my design process by keeping it centered on my own knowledge and experiences.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>regularly and critically review my design work in progress and the way that it was developed.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>define parts of the problem or solution based on a boundary that describes relationships between those parts.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>follow a pre-defined sequence of steps to move from problem to solution, using a trial and error approach to solve each part of the problem.</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

**Design Decision Making:**

The following statements relate to how you make and justify or support decisions that you make in the course of designing solutions to complex software problems. Please indicate your level of agreement or disagreement with the following statements, using the scale 1) Strongly Disagree, 2) Disagree, 3) Not sure, 4) Agree, 5) Strongly Agree.
Each item below completes the following sentence: “When I am working on a complex software design problem, decisions I make about the problem and how to solve it are...”

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>confident, reflecting my awareness of the consequences of my decisions.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>solutions that follow directly from the information available at that time.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strongly grounded in my personal experience.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rarely final and are open to alternatives and changes as the design progresses.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tentative and uncertain until they are written as executable code and evaluated.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>directed towards meeting the explicitly stated requirements as I understand them.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>often use “soft” or subjective information as a means for building my understanding of the purposes and goals for the system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>closed to change or modification once decided.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>based on first principles or proven best practices of the problem and application domains.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>creative and often do not logically follow from the information that is available.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>confidently made when I have numerical information available.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>oriented towards the overall purposes and goals for the system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

Exit/Feedback Questionnaire

Overview and Instructions

The purpose of this questionnaire is to gather feedback from you regarding the Principles, Patterns, and Process Framework you were introduced to in this study and about your experience as a participant in this research study. No personal or identifiable information will be collected as part of this feedback.

This survey consists of three groups of statements about different aspects of your participation and an area where you may write any comments, suggestions, or other feedback.

The Principles, Patterns, and Process Framework

Please indicate your level of agreement or disagreement with the following statements about the Principles, Patterns, and Process Framework itself, using the scale 1) Strongly Disagree, 2) Disagree, 3) Not sure, 4) Agree, 5) Strongly Agree.

<table>
<thead>
<tr>
<th>Statement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>I was able to understand the Principles.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I was able to relate the Principles to designing software.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Pattern structure used to document the Principles was logical and clear.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Pattern structure helped me relate the Principles to each other.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Pattern structure enabled me to apply the Principles to design problems.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The steps in the Process are logically ordered.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The steps in the Process are clearly stated.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I was able to use the Process to guide the decisions I made while working on design problems.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Principles, Patterns, and Process Framework Website

249
Please indicate your level of agreement or disagreement with the following statements about
the Principles, Patterns, and Process Framework website, using the scale 1) Strongly Disagree,
2) Disagree, 3) Not sure, 4) Agree, 5) Strongly Agree.

<table>
<thead>
<tr>
<th>Statement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Principles, Patterns, and Process Framework website is logically organized and easy to navigate.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The Principles, Patterns, and Process Framework website helped me learn how to use the Framework to solve design problems.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The Principles, Patterns, and Process Framework website should be extended to include other software design and engineering principles.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The Principles, Patterns, and Process Framework website should include more examples.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The tutorials provided on the website clearly illustrated how to use the Framework to help solve design problems.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Design Problems

Please indicate your level of agreement or disagreement with the following statements about
the design problems used in this research study, using the scale 1) Strongly Disagree, 2) Disagree,
3) Not sure, 4) Agree, 5) Strongly Agree.

<table>
<thead>
<tr>
<th>Statement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>The design problems I was given were all about the same level of difficulty.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The design problems I was given were too difficult for someone of my skill level and experience.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>I was familiar with all of the concepts and terminology used in the design problems I was given.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The design problems were clearly stated and provided enough information for me to complete the requested task(s).</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>I needed more time to complete the problems.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>I am satisfied that the solutions I was able to develop for the problems were correct and met and requirements or goals stated in the problems.</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Comments and Suggestions

Please use the space below to write any additional comments or suggestions about the
Principles, Patterns, and Process Framework and website and/or this research study. You may use the back of this page if necessary. Please do not include any personally identifiable information in your comments.

250
Appendix E

Protocol Coding Instructions

E.1 Overview

One objective of this research study is to investigate how and why software designers make decisions during the process of developing a solution for a complex design problem. For the purpose of this work, we define a decision as a set of one or more utterances related by a common subject, object, or expressed intent. In the vocabulary of verbal protocol coding, a decision constitutes a segment of the protocol.

Subjects in the study will be given one hour to work on a design problem while “thinking aloud” while being audio and video recorded. These recordings will be transcribed and separated into approximately fifteen minute blocks, based on naturally-occurring pauses and/or significant changes in focus. You will be given randomized sets of these blocks to segment and code at regular intervals. Synchronized transcriptions of the audio recordings will be provided. The recordings and associated transcripts will be packaged for use in the Atlas.ti qualitative analysis software system as individual Hermeneutic Units (HU), complete with the codes defined in Section 3 below.

The HU data will be delivered to you on a USB flash drive. To keep data transfers manageable and to help prevent overloading, you will be given twelve to fifteen HUs at a time (about three to four hours of recorded material. You should save your work on the flash drive, within the Atlas HU data structure. When you complete the segmentation and coding of the HUs on a flash drive, it will be exchanged for another flash drive containing a new set of HUs to be coded.

E.2 Decision Segmentation

Segmentation of the protocols will be the initial task. For this project, a decision as a set of one or more utterances related by a common subject, object, or expressed intent, and constitutes a segment of the protocol. An utterance will usually be a spoken phrase or sentence, but may also be expressions made with pen-and-paper, e.g., hand-written notes, sketches, etc., or on a computer, e.g., typed noted, manipulation of a graphical design model, etc.

For example, a subject might say something like “I need an object like this...” and add a graphical element to a design model using a computer design tool. This expresses a decision
to make some change in the existing design to accommodate new information. Alternatively, a subject might utter “I need to figure out what this requirement means...” Again, we will consider this to be a decision: the subject has determined that more information is needed and where to begin looking for that information. A subject may also make a decision to stop working or otherwise abruptly change how he/she is approaching the problem, i.e., “I’m confused; I don’t know where to go from here...”

**E.3 Decision Coding**

Once you have segmented the decisions in a protocol, each decision must be coded to characterize the decision. For this project, you will apply four code variables to each decision. The four variables correspond to four general categories of information: the level of abstraction or detail the decision involved, the type of decision made, why the decision was made, and the subject or object of the decision. Coding variables have been defined for the first three categories and will be discussed in more detail later in this section. The variables for the fourth category will depend upon the particular problem the subject is working on and the names or terms used by the subject, and so cannot be determined ahead of time.

We also include a general, non-design set of coding variables, shown in Table E.1 below. These variables should be applied, where appropriate, in place of the normal four-variable coding.

**Table E.1: Non-design Coding**

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unusable Fragment</td>
<td>Any coded fragment irrelevant to the protocol</td>
<td>“I need to take a short break”</td>
</tr>
<tr>
<td>(N-Un)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confusion</td>
<td>A fragment that explicitly states that the subject is stuck or confused and</td>
<td>“I wish I could understand this”</td>
</tr>
<tr>
<td>(N-Conf)</td>
<td>does not know how to proceed</td>
<td></td>
</tr>
<tr>
<td>Non-verbalized</td>
<td>A fragment of the recorded protocol where the subject is engaged in working</td>
<td></td>
</tr>
<tr>
<td>thinking</td>
<td>on the problem but is not verbalizing her/his thoughts</td>
<td></td>
</tr>
<tr>
<td>(N-NV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop</td>
<td>Subject has decided to stop working on an incomplete solution</td>
<td></td>
</tr>
<tr>
<td>(N-Stop)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td>Subject has decided that the solution is as complete as he/she can make it</td>
<td></td>
</tr>
<tr>
<td>(N-Comp)</td>
<td>and will stop working on it</td>
<td></td>
</tr>
</tbody>
</table>

The variables for *decision level coding* are listed and defined in Table E.2. Briefly, these variables describe a range of abstraction or detail level from abstract conceptual ideation to
specific, concrete details of the proposed design. The table lists the definitions for each of the variables and provides examples of utterances that could help to identify which variable would be the appropriate code for a segment.

Table E.2: Decision Level Coding

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual (L-Con)</td>
<td>Understanding and characterizing the problem, focusing on concepts and ideas that help build understanding</td>
<td>“This part of the system is like a ...” “The system fits into the ... part of the user’s workflow.”</td>
</tr>
<tr>
<td>Requirements (L-Req)</td>
<td>Defining, analyzing, and refining stated requirements and eliciting additional requirements</td>
<td>“The problem statement says...” “The system must handle 50 concurrent users without a noticeable performance loss, right?” “Users must be able to view reports based on their access level”</td>
</tr>
<tr>
<td>System (L-Sys)</td>
<td>Describing and refining global characteristics of the system when viewed as a whole, complete unit</td>
<td>“The system is responsible for ... across the company” “This application will integrate on-site point-of-sale with the company’s online e-commerce website”</td>
</tr>
<tr>
<td>Component (L-Com)</td>
<td>Distinguishing and developing major component parts of the system</td>
<td>“We’re going to need a database and a PHP-enabled web server” “The hourly billing part of the system must correctly pass employees’ billed time to the payroll part of the system”</td>
</tr>
<tr>
<td>Detail (L-Det)</td>
<td>Defining and evaluating low-level, concrete details necessary to implement the proposed solution</td>
<td>“The data table must have a field for ...” “Users’ configuration information should be maintained in a local file”</td>
</tr>
<tr>
<td>Interaction (L-Int)</td>
<td>Defining, describing, and analyzing interactions between the system and various agents that use or are used by the system. These agents include human users as well as software-based entities that interact with the system</td>
<td>“Users will be required to log into the system using a secure user ID and password” “The system must transmit correctly formatted data files to the IRS and the state Department of revenue”</td>
</tr>
<tr>
<td>Environment (L-Env)</td>
<td>Understanding and characterizing the systems execution environment(s)</td>
<td>“I think Linux will be the best choice for the web servers” “Remote users will be able to access the system through virtual private networking”</td>
</tr>
</tbody>
</table>

Table E.3 lists the coding variables for the decision type category, again with definitions and examples. There are two pairs of variables that are subtly different. Decomposition and Differentiation both involve separating a large problem or entity into smaller parts. Similarly, Composition and Integration involve bringing parts of a solution together. Decomposition is similar to factoring in a mathematical problem: the problem of finding a solution is separated into finding solutions to smaller and simpler problems that can then be simply recombined (Composed) into the solution for the larger problem.

Differentiation, on the other hand, is focused on the relationships between the separated
parts, usually through the definitions of the parts’ boundaries, interactions, and contributions to the larger problem or entity. The separation of the parts is driven by the identification of different aspects, responsibilities, etc. within the larger problem or entity. Similarly, an integration decision involves bringing related parts together in a consistent, structural manner.

The coding variables for the decision segment rationale category are shown in Table E.4 with definitions and example utterances. These variables will be used to identify what justification or basis was used for making a design decision.

The final coding category is the subject and/or object of the decision. As noted earlier, the variables in this category will depend upon the particular problem the subject is working on in the protocol. The problems that will be given involve different types of software applications, so there will be differences in the terms used in the problem and application domains. Additionally, as the subject begins developing her/his solution, names or labels will be defined to identify parts of the solution, and these names will vary from subject to subject. Mapping these names back to the generic solution parts will be done during the analysis of the codings, so you do not need to be concerned with that task. It will be sufficient for you to simply code whatever name or label the subject chooses to use in the protocol.

We have attempted to create a coding scheme that is as complete as possible, and ideally, each identified decision will be characterized by four coding variables. However, there may be situations when none of the code variables in a particular category may apply. If you encounter such a case, you may add a variable to the appropriate category to characterize the situation more clearly. If you do add a variable, please notify me via email at david_wright@ncsu.edu so that I can record the change and pass it along to the other coder. You may also use either of these contacts to address any other questions or concerns about this work.
### Table E.3: Decision Type Coding

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition (T-Dec)</td>
<td>Separate a problem into smaller parts with simpler solutions that can then be recombined to solve the larger problem</td>
<td>“This is a recursive-type problem”  “An appointment contains at least a date, a time, a place, and a title or subject”  “What characteristics or functions distinguish different types of users?”</td>
</tr>
<tr>
<td>Differentiation (T-Dif)</td>
<td>Distinguish between different parts or elements of a problem or system, usually through the definition of boundaries or other inter-element relationships</td>
<td>“There are three major parts of this system that have to work together”  “Model-View-Controller”  “Managers, sales staff, and the accounting department have different ideas about what the system needs to do”</td>
</tr>
<tr>
<td>Abstraction (T-Abs)</td>
<td>Generalization of problems or design elements, used to classify, categorize, or simplify an aspect of the problem or design</td>
<td>“This problem is an optimization problem”  “What do the different kinds of users have in common?”  “The company sells a variety of products and offers other services on a pay-per-instance basis”</td>
</tr>
<tr>
<td>Instantiation (T-Ins)</td>
<td>Assignment of specific details and/or unique attributes to make a particular design element more concrete or specific</td>
<td>“A customer is identified in the system by a unique identification number”  “The creation of an artifact such as a diagram, code snippet, documentation, etc.”</td>
</tr>
<tr>
<td>Composition (T-Com)</td>
<td>Combining solutions to smaller problems to form a solution to a larger problem; the inverse of decomposition</td>
<td>“A Customer object will extend the User class and implement the RetailBuyer and Commenter interfaces”  “To display weekly schedule, we first search for the appointments within that time, then use the client ID to look up the client information”</td>
</tr>
<tr>
<td>Integration (T-Int)</td>
<td>Similar to composition, but using a larger-scale perspective structuring a solution (or set of solutions) within the context of the system as a whole (or of a larger part of the system)</td>
<td>“The new system will use the existing database system, which will be modified to store the additional information needed”  “Our development, deployment, and support processes should work smoothly with ABC Company’s current business processes to avoid disrupting the company’s finances”</td>
</tr>
<tr>
<td>Postponement (T-Pos)</td>
<td>The need for a particular design decision has been identified but deferred in favor of another (usually easier or less risky) decision or because there is not enough information to make an informed decision at this time</td>
<td>“I will decide on this part of the design after I meet with the customer”  “I’m not sure which solution will be better for this part of the system, but I know that this other part can be reliably resolved, so I will do that now and come back to the first part later”</td>
</tr>
<tr>
<td>Proposal (T-Pro)</td>
<td>Suggesting or hypothesizing a particular solution or solution fragment</td>
<td>“Based on the customer’s needs and budget, we should go with a LAMP-based solution”  “I think this will work...”</td>
</tr>
<tr>
<td>Information Gathering (T-IG)</td>
<td>Seeking out additional information not included in the problem statement in support of the design process</td>
<td>Asking questions of the observer/customer, Online research, etc.</td>
</tr>
<tr>
<td>Evaluation (T-Eva)</td>
<td>Testing a solution or solution fragment in the context of the problem it is proposed to solve</td>
<td>“I know this code works in other situations, but I need to be sure it works here”  “With these values as inputs, the program should produce this output while generating an error for values larger than 1000 or smaller than 0”</td>
</tr>
<tr>
<td>Code</td>
<td>Definition</td>
<td>Examples</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>Requirements or Problem Statement (R-RPS)</strong></td>
<td>Information explicitly stated in or logically deducible from the given problem statement and/or requirements is used as the basis for a decision</td>
<td>“The problem statement says that this condition must always be true, so the system must enforce this as an input validation rule” “The requirements state...”</td>
</tr>
<tr>
<td><strong>Application Knowledge (R-AK)</strong></td>
<td>Recall and use of previously gained knowledge about the application or the environment in which the designed artifact is to be used, e.g., database, web server, user interface, etc.</td>
<td>“In the database class I took last year, we learned how to optimize a database using normal forms” “I have configured Apache web servers before, and I know that this is the correct way to do...”</td>
</tr>
<tr>
<td><strong>Domain Knowledge (R-DK)</strong></td>
<td>Recall and use of knowledge about the problem domain, e.g., business processes or rules, accounting procedures, regulations, etc.</td>
<td>“In a part-time job I held last summer, I did bookkeeping for a small company and learned basic accounting procedures” “According to state statutes, the system must calculate and charge North Carolina state sales tax on any purchase shipped to a North Carolina address”</td>
</tr>
<tr>
<td><strong>Best Practices or Design Principles (R-BP)</strong></td>
<td>Specific, documented principles or practices applicable to the problem and context justify a particular choice</td>
<td>“One of the rule of Extreme Programming is that we should only work 40 hours per week on average” “I can implement the Abstract Factory pattern to create different instances of this class at runtime”</td>
</tr>
<tr>
<td><strong>Formal or Experimental Analysis (R-FEA)</strong></td>
<td>The decision is a result of a logical analysis of alternatives or a mathematical proof</td>
<td>“We tried 3 different implementations of this algorithm with the same set of input data; number 2 was the fastest, but used a lot of memory; number 3 was very slow and used even more memory than number 2; number 1 was slower than 2, faster than 3, and used less memory than either.”</td>
</tr>
<tr>
<td><strong>External Reference Material (R-ER)</strong></td>
<td>Look-up and use of material outside the given problem statement and the subject’s own direct knowledge and experience</td>
<td>“I found this code online and it should work with a little modification” “According to the MySQL documentation, we should do...”</td>
</tr>
<tr>
<td><strong>Personal experience (R-PE)</strong></td>
<td>Decisions based on generic, non-specific personal experience, guessing, etc.</td>
<td>“I’ve always done it this way and never had a problem before...”</td>
</tr>
<tr>
<td><strong>Unknown or Indeterminable (R-Unk)</strong></td>
<td>No clear indication of the justification for a decision</td>
<td>Nothing to indicate why a particular decision or choice was made</td>
</tr>
</tbody>
</table>