ABSTRACT

LONG, DANIEL ESLEY. System Excess Placement for Improving Lifecycle Value. (Under the direction of Dr. Scott Ferguson.)

The objective of this research is understanding, modeling, and evaluating the use of strategic overdesign (excess) as a method for minimizing the cost of system change to maximize system lifecycle value. This research is necessary because the design and construction of modern complex engineered systems is costly, and these systems operate in a context that changes over time. Reducing (and ideally minimizing) the cost of executing system adaptations is therefore advantageous. Prior research provides guidance for how system changeability can be supported by encapsulating functionality within modules, but little research has been dedicated to optimal design variable (or component sizing) selection to support future system changes. The specific goal of this research is addressing the following question: ***What placement of reserve margin within a given architecture provides optimal flexibility to mitigate future uncertainty for a system?***

Strategic Excess Placement for Improving Lifecycle Value

by
Daniel Esley Long

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

Aerospace Engineering

Raleigh, North Carolina

2020

APPROVED BY:

_____	_____
Dr. Scott Ferguson Committee Chair	Dr. Julie Ivy
_____	_____
Dr. Andre Mazzoleni	Dr. Gregory Buckner

BIOGRAPHY

Daniel Long is a PhD candidate studying Aerospace Engineering at North Carolina State University. His research uses data science for studying how system changeability and value may be improved by including excess in the initially fielded system. Daniel received his B.S. in 2006 from North Carolina State University in Nuclear Engineering and his MS degree in Aerospace Engineering from North Carolina State University in 2014. Daniel has worked for Progress Energy as a Fuel Supply and Nuclear Core Design engineer and for NASA Langley as an intern in the Space Missions Analysis Branch. Daniel has also completed the International Space University's Space Studies Program and a fellowship at The Data Incubator.

ACKNOWLEDGMENTS

The author would like to acknowledge support from the American Public Power Association as part of a Demonstration of Energy and Efficiency Developments Grant, the U.S. Department of Energy through the Cities Leading through Energy Analysis and Planning Award, and the Garrett Fellowship.

I would like to thank and to express my most sincere gratitude to my adviser Dr. Scott Ferguson for his tireless effort in guiding my research, his help trimming and focusing drafts that were always far too long, and helping me grow as a researcher and engineer. I would not be where I am today without his help. I would also like to thank the member of my committee Dr. Gregory Buckner, Dr. Andre Mazzoleni, and Dr. Julie Ivy for both their insightful feedback on my research and for the roles they have played in teaching and guiding me throughout my academic career.

I would like to thank my former and current fellow students in the System Design and Optimization Laboratory. You have each helped me during my time as a student by teaching, sharing, and supporting me when I needed it.

Finally, I would not be where I am today without the patience, support, and love from friends, family, and pets. You all kept me going! I am especially for the support from my partner April Cash who has been patient and supportive when work hours grew long.

Thank you all!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

1.1 Motivation

The scale, interconnectivity, and interdisciplinary nature of new systems is making the design of complex engineered systems (CES) increasingly difficult. There is general acknowledgement in prior research that existing design methods fall short when creating these systems [1–4]. Symptoms of this shortfall are seen in delayed schedules and increased costs for major engineering projects. Prominent examples include the F-35 Joint Strike Fighter (JSF) and the James Webb Space Telescope (JWST).

The JSF has experienced an increased per-unit-cost of 69% with an acquisition time increase of 35% [5]. Current estimates put the total cost of the 55 year lifetime at $1.2 trillion - of which $428 billion is for acquisition costs - making it the most expensive weapons system ever developed [6]. The James Webb Space Telescope had to be entirely rebased in 2011, increasing the total cost of the system from $3.5B to $8B, and postponing the launch date from 2011 to 2018 [7]. Additional assembly and integration difficulties have pushed the expected launch date to March 2021 - adding an additional $1.7 billion – resulting in a total cost of $9.7 billion [8]. This is almost an order of magnitude greater than the first estimate in 1996 [9]. The complexity inherent in these systems is one of the primary challenges in their design [10] and the cost and schedule overruns have consequences for subsequently designed systems.

One consequence is that **CES are expected to remain in service for ever longer periods of time since replacement costs are highly uncertain**. Evidence of this exists in the nuclear power industry where plants initially awarded 40 year operating licenses are being renewed for 20 and 40 additional years [11]. Aircraft like the B-52 and C-130, placed in service

in the 1950's, are expected to remain in service for 80-100 years [12,13]. Planning for long operational periods creates a feedback loop that further increases the complexity of CES design.

As operational periods are extended, each marginal increase offers diminishing returns. Systems are designed within a specific context comprised of a set of available technologies, existing customer preferences, and anticipated interactions with other systems [14]. This context drives initial system requirements which in turn affect how the system is designed. As time passes following system deployment, **the context in which a system operates diverges from the one in which it was designed**. Extended operational periods do allow additional benefit to accrue from the initial investment, but those benefits diminish as the context changes unless a system can be modified to better fit the new context [15].

Design researchers and industry practitioners have begun recognizing the importance of designing complex engineered systems where the cost and effort of modification after production is reduced so that premature system obsolescence can be prevented [16,17]. A familiar example is the regular software updates on modern computers and phones. The same technique is seen in hardware, as exemplified by block upgrades for modern military aircraft or in the ability to upgrade desktop computer components piecemeal. ***The need to constantly evolve leads designers away from designs which are optimal in the current context and towards the robust design of a "structure with good bones"*** [18] ***to support future design changes.***

1.2 The Phenomenon of Change Propagation

Change propagation is a primary contributor to the cost and complexity associated with modifying a system. Change propagates when the modification of a single component results in the modification of other components. For example, replacing an old desktop GPU with a more powerful model likely requires additional electrical power. This may require upgrading the

power supply, increasing the modification costs by the price of the new power supply. Assessing change propagation therefore plays a central role in designing changeable systems.

Existing change propagation research builds on system architecture modeling. Ulrich [19] provides an early and concise description of architecture as: 1) the arrangement of functional elements (the exchange or conversion of signals, materials, forces, and energies), 2) mapping functional elements to physical components, and 3) the specification of interfaces between interacting physical components. Product architecture is important to change propagation because it is a framework for modeling component coupling. Ulrich describes two components as coupled "if a change made to one component requires a change to the other component in order for the overall product to work correctly." Prior research has found that a change in one component can result in the need for change in many other components due to the transmission of change along chains of coupled components [20]. Change propagation drastically increases the number of modifications, and therefore the cost, required to support a single desired change.

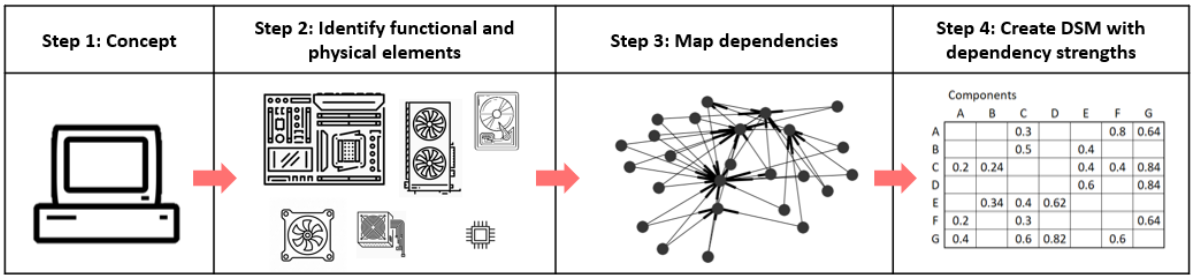


Figure 1.1: The four steps of architecture-based change propagation

A large branch of change propagation research uses the general approach shown in Figure 1.1. The process begins with the identification of a system's functions, functions are then mapped to physical elements, and the interfaces and design dependences between elements are captured. A network representation of the system is then created where components are represented as nodes and the dependencies connecting them are represented as edges connecting the nodes. The system graph is then converted to an adjacency matrix referred to as a Design

Structure Matrix (DSM). Additional information is frequently captured in addition such as: the strength of the dependencies (represented as edge weights), the links between components and consumer preferences, the number of anticipated changes of a component [21], or other networks of interest like the social network connecting the designing engineers [22].

The system architecture networks are then for creating metrics that capture an aspect of changeability. The goal of these approaches is striking an appropriate balance between modularity and integrality. Modularity is defined as "*a one-to-one mapping from functional elements . . . to the physical components of the product, and specifies de-coupled interfaces between components*" [19] and incurs additional costs from the introduction of new interfaces between modules. Integrality is defined as components having one-to-many or many-to-many mappings to functional elements and/or other components. System architecture approaches are valuable for identifying tightly coupled components. Strategies for coping with component coupling are: identifying and avoiding them when making modifications [23], decoupling them as much as possible from the system [24], and clustering the components into a single physical element with minimal dependencies within the remainder of the system [25].

The above approaches have two major limitations in common. First, multiple independent modifications are not captured. For example, a common phenomenon with military aircraft is that they become heavier as new capabilities are added. New sensors, computers, ordinance, and safety systems are incorporated as the system ages. An initial analysis for a new aircraft using existing techniques would not identify weight as an issue. If the analyses are repeated after each modification, they would still be unlikely identify the problem until a modification violates a weight related requirement. At that point, change propagation analyses would show dramatic increase in change propagation to compensate for the added weight.

A second limitation is that designers have little quantitative guidance regarding component sizing during system design. This information may be implicitly captured in the supplemental information as reduced change propagation probabilities or weaker dependency values, but a direct link is not made.

The common cause of these two limitations is that component parameters, system parameters, and their associated requirements are not explicitly captured by these approaches. Overcoming these limitations therefore requires the collection and explicit modeling of these phenomena.

1.3 Design Margin and Excess

Research into *excess* offers a new perspective on collecting and modeling the effects of parameter values and system requirements. The specification of design margin is an additional degree-of-freedom available to designers not well modeled by existing change literature. Eckert et al. call design margins "a hidden issue in the industry" [16]. Eckert et al. suggest that "the key issue in predicting change propagation within complex engineering systems is in understanding the tolerance margins of the key parameters relating to the major systems" [20]. E*xcess* is defined as margin that is "the value over, and above, any allowances for uncertainties" [16]. Figure 1.2 illustrates the relationships between margin, buffer, and *excess*. *Excess* is distinct from buffer as a type of margin although reductions in uncertainty from new data or more rigorous analysis can convert buffer into *excess*.

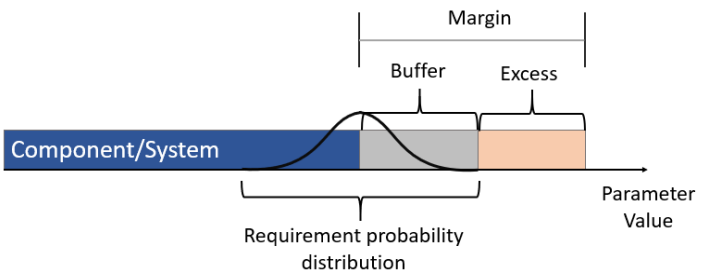


Figure 1.2: Notional example showing relationships between system parameters, margin, requirements, and *excess* [16]

Existing change propagation research may capture excess in aggregate via a dependency strength metric, but it is not modeled explicitly. The concept of e*xcess* is a promising avenue for forecasting the ramifications of multiple, future system changes. The goal of optimally placing *excess* therefore becomes a foundational challenge, as margins often mean larger up-front costs. Realizing this goal requires fully modeling the consequences of change propagation throughout a system's life. This research studies the optimal allocation of *excess* by considering the costs and benefits.

1.4 Research Plan Overview

Intentionally incorporating changeability typically involves a cost or performance penalty in the present so that the future costs are reduced. This has been demonstrated in product platforming [26] and real options in systems literature [27]. The difficulty when planning for future events is that the timing, impact, and future system states are all unknown in the present. The primary research question is therefore: **What placement of excess within a given architecture provides optimal changeability for uncertain future states?**

One important aspect of this question is understanding how excess enables changeability. If changeability is defined as a reduction in future costs, then how does excess reduce those costs? ***The hypothesis in this research is that excess reduces the cost of future changes by inhibiting change and change propagation***. Intuitively, if a power supply is designed to supply

exactly the current load required when the system is initially fielded, then any future increase in power demand will require replacement of that power supply. The absence of reserve margin in a power supply therefore increases the cost of future changes. The following research questions guide the research process in the remainder of this document.

Research Question 1) What system design lessons can be learned from qualitative evidence linking the presence/absence of excess and system lifecycle value?

This question is addressed by studying two military aircraft: the B-52 Stratofortress and the F/A-18 Hornet. Chapter 3 discusses the historical context in which these systems were developed, how that context influenced the inclusion of *excess* of these systems, and what the repercussions of the degree of excess inclusion were on the system lifecycles.

Research Question 2) What system design lessons can be learned from quantitative evidence linking the presence/absence of excess and system lifecycle value?

Addressing Research Question 2, Chapter 4 uses historical records for computer requirements for video games for a quantitative study on the impact of excess. Game publishers must clearly communicate requirements about the necessary level of computer performance required to play their games. These requirements are captured in archived magazines and databases with records dating back several decades. Analysis of this data provides insight into how requirements for computers changed over time. Additionally, industry publications like *PC Gamer* magazine offer component suggestions for building gaming computers. These guides provide contemporaneous records for tiers of performance with varying levels of excess built into each. Combining the suggested computer builds with subsequent video games requirements the system coped with provides a quantitative examination of the value of excess. Additionally, Chapter 4 assesses the value associated with strategic excess (excess added to a single

component) and evaluates the appropriate level of excess given the coevolution between hardware improvements and game requirements.

Research Question 3) What can system design lessons can be learned by extending existing change propagation methods by accounting for excess and how satisfactory is the extended tool for modeling excess?

Chapter 5 combines change propagation research with *excess* by extending existing a change propagation method to incorporate the impacts of multiple changes. The focus of this section is the extension of CPM [23] for supporting an analysis of multiple changes, rather than just a single change. CPM is an architecture-based change propagation methodology that captures dependencies between components and the strengths of those dependences to provide designers guidance regarding which are likely to spread change when modified. The method is extended by assuming that the dependencies linking components are made stronger with the consumption of component excess. By sampling stochastic system lifecycle trajectories (the sequence of initiating component selection with subsequent affects change propagation), we study how the patterns of connections and weights of dependencies impact lifecycle performance.

Research Question 4) What phenomenon must be included for modeling the effects of excess and how can they be combined for creating a holistic assessment of excess location and degree on system lifecycle value?

The final research question (Chapter 6) is the culmination of research done for previous tasks and is the development of a generic modeling method that can be applied for quantitatively assessing excess placement within a system. This model incorporates each aspect of the system design process necessary for comparing lifecycle value for variations of excess placement and

quantity. This includes linking system design variable specification with cost and initial system performance, modeling design dependencies to allow for procedurally driven change propagation and enabling system lifecycle trajectory sampling by modeling time dependent exogenous variables. Using system design space sampling, the results of the final research question enables designers to search the space of initial system design and lifecycle decision making to estimate the value of excess placement within a system.

1.5 Dissertation Outline

This dissertation is divided into 7 chapters. Chapter 1 provided the motivation and the four research questions addressed. Chapter 2 provides a background for engineering change, change propagation, and excess research sufficient to ground the reader in the concepts required to answer the research questions. Chapter 3 is a qualitative examination of two military aircraft lifecycles providing supporting evidence for excess enabling system changeability by limiting change propagation. Chapter 4 is a quantitative study evaluating excess using contemporaneously suggested PC computer builds compared to the requirements of subsequent video game releases. Chapter 5 details an existing system-level change propagation tool including and extends it to incorporate the use of excess within the system to inhibit change propagation. Chapter 6 details a novel detailed component level methodology for explicitly capturing the affects and value of incorporating excess in a system. Finally, Chapter 7 concludes the dissertation with a discussion of how the research conducted has addressed the research questions and what avenues future research might follow.

Chapter 2: Background

2.1 Introduction

The background section has two aims. First it provides context for the research questions with a discussion of engineering design focusing on change and flexibility/changeability literature. Second it provides detailed accounts of the concepts and methods from prior research used in subsequent chapters. These two aims are split among three subsections:

- Engineering Design Process and Engineering Change Context
- Component-Level Change-Importance Metrics
- Excess and Margin Research

2.2 Engineering Design Process and Engineering Change Context

The engineering design process encompasses all the steps necessary to realize an engineered system from need identification to system retirement and disposal. “In reductio ad absurdum, engineering design involves only two steps: (1) determine all possible design options and (2) choose the best one” [28]. These two steps are overly simplistic for practical use, but they do provide a basis from which to understand the goal design process with one addendum. Engineering design, like academic research, is constrained by limited resources (e.g. time, money, effort). The key to successful design is therefore to maximize value added from the expenditure of those limited resources. Applying these design principles to changeability leads to three required assessments. These are:

- What is flexibility?
- What design features add to system flexibility and how are those feature measured?
- How can one compare those options and choose the “best one”?

Addressing these three questions is the focus of this section.

2.3 What is flexibility?

To begin let us consider what changeability is in the context of an engineered system. This research adopts the terminology proposed by Frick and Schulz [14] who use **changeability** as an umbrella term for a system's ability to change after being placed into service. Changeability has four subcategories:

- *Robustness* – the ability to remain insensitive to changing environments
- *Agility* – the ability to change quickly
- *Adaptability* – the ability for the system to change itself internally
- *Flexibility* – the ability for the system to be changed easily by an external source

Saleh et al. [4] compares the state of changeability research to safety research decades ago as "vague and difficult to improve, yet critical to competitiveness". As discussed in the introduction the general aim is to enable a system to adapt to changes in its context to get more value from the resources expended during the design and production of that system. Prior research provides many alternative flexibility ontologies and approaches. Hamraz et al. [29] examined engineering change research and found 73 papers related to "concepts to prevent or to ease the implementation of engineering changes before they occur". Among these papers are several surveys of flexibility, including a framework that separates flexibility into five phases by the kinds of activities supported [30], a survey of reconfigurability and flexibility [31], and a multidisciplinary literature review of flexibility related research [4]. Each provide a more comprehensive overview of the subject.

This research primarily focuses on the flexibility aspect of changeability, except for Chapter 4 which assesses the how excess enables a system to satisfy changing and unknown future requirements.

2.4 What design features add to system flexibility and how are those features measured?

Features that inhibit change propagation increase a system's changeability. Saleh describes the incorporation of flexibility more as "the inverse of a system's impedance to change." [4] A starting point for how this may be accomplished is provided by Suh [32] who argues that a flexible system allows for new design parameter selection to match time varying functional requirements. Frick and Schulz [15] add that those changes should require minimal effort. Flexibility is most often added by the use of modularity. This section therefore begins with a brief overview of modularity research. The section continues with a discussion of aspects of modularity research also useful in excess research and concludes with a deep dive on Design for Variety, which exemplifies many prototypical elements used by other modularity research.

Like changeability the precise definition of modularity has been widely debated. Ulrich proposed that systems fall on a scale between fully modular (characterized by one-to-one mapping between functions and components) and integral (characterized by non-one-to-one mappings) [19]. Engineered systems fall between these two extremes, and determining where on the spectrum a system is classified is non-trivial. Hölttä-Otto [33] provides an overview of 13 methods for measuring system-level modularity and found that tested methods primarily focus on measures of component similarity and/or component coupling. Most were found unsatisfactory due to inconsistency or the inability to account for the presence of a system bus.

An early study by Gershenson et al. [34] compared existing modularity metrics with modularity ratings from study participants (undergraduate and graduate students, practicing engineers, design engineers, and design researchers with an interest in modularity). They found no statistical correlation between ratings and metrics. After an extensive literature review three fundamental elements of modularity were identified:

- the independence of a module's components from external components
- the similarity of components in a module with respect to their life-cycle processes,
- and the absence of similarities to external components.

Each of these elements drives a system to reduce the scope of modifications required for supporting a desired change by placing elements likely to change together and limiting connections from that module to other parts of the system.

Modularity research has been more successful with component-level metrics. These metrics do not seek to provide an overall assessment of system modularity, but instead identify components and interactions that may benefit from added modularity. That information can then be used by designers to improve system designs.

Product platforming research is an example of successful incorporation of these metrics with the design process. Product platforming focuses on the same concepts as flexibility (reducing the effort of changes) but at the product design stage instead of for a fielded system. A review of product platforming may found in Jaio et al. [35] with a more recent integrated framework in Simpson et al. [36]. The premise is that a firm creates a core design with largely common elements and then leverages that core design with segment specific elements. The key is selecting which components and functionality should be kept common and which should be differentiated. This is exactly what component level metrics enable. The metrics provide information about component interconnectedness and the degree of change required for each market segment. The example given in Simpson et al. [36] is a firm that specializes in unmanned ground vehicles (UGVs) shown in Figure 2.1. Some components are entirely different for each architecture (like the arm and grippers) and are modularized to minimize connection to the rest

of the design. Other components like the batteries are more standard and are kept common with some allowance for scaling.

(a) Bombot http://www.defensetech.org (b) Talon http://www.foster-miller.com (c) Packbot http://www.irobot.com (d) RONS http://www.globalsecurity.org

Figure 2.1: UGVs with different market segments displaying commonality and differentiation [36]

There are several component-level approaches found in existing literature including Change Propagation Method [23], a network based approach [37], Change Propagation Index [38], and Design for Variety [24]. These approaches were conceived for supporting modularity inclusion or change-path selection, but they also provide guidance for possible excess placement. Each of these methods, except the network-based approach, is discussed in detail in the following subsection due to their importance in later chapters.

A final noteworthy study was conducted by Tilstra et al. [39] which sought to address how flexibility is added to existing commercial products. The researchers examined patents from 250 products with flexibility related terminology in their descriptions and empirically studied other commercial products. These were analyzed to find common design features. The result is a 24-point list of guidelines for designing flexible systems with 5 principles: modularity, parts reduction, spatial, interface decoupling, and adjustability. Many of the guidelines focus on simplicity (reducing and standardizing parts) or interface management (decoupling, providing room on exterior surfaces, providing free interfaces, etc…) with two ("controlling the tuning of design parameters" and "providing the capability for excess energy storage or importation")

hinting at the idea of *excess*. The guidelines are useful heuristics but do not lend themselves to quantitative analysis.

2.5 How can design options be compared to find a "best" one?

The research conducted in Chapters 4 and 6 adhere to the Decision Based Design (DBD) framework proposed by Hazelrigg [28,40] for selecting which is best. The goal of DBD is providing a rational framework for design decision making based on the principles of a systems level valuation. Figure 2.2 outlines the aspects of DBD along with the flow of the process among those aspects.

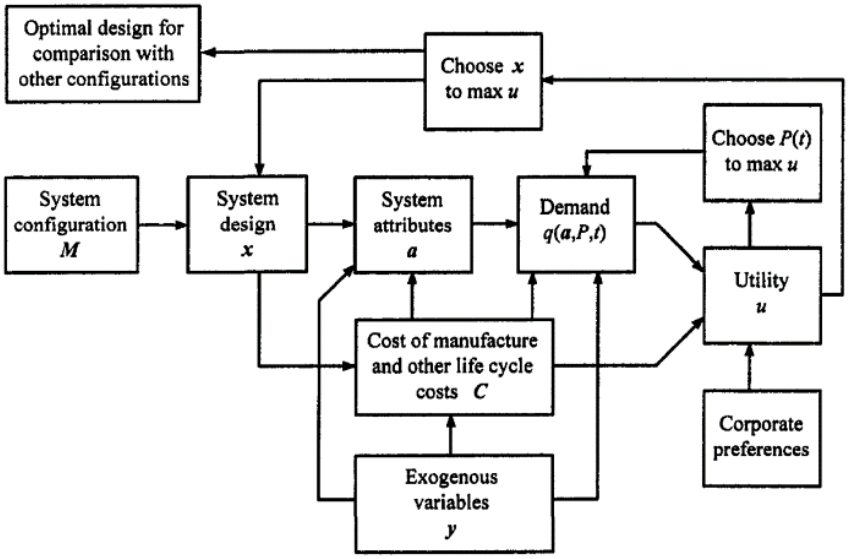


Figure 2.2 A Decision Based Design Approach [28]

The premise is that system configurations (the functional and physical arrangement of a system) should be compared by their optimized expected utility. The utility is derived from a combination of system demand (itself a function of attributes (a), costs (C), exogenous variables (y), and price (P)), costs, and corporate preferences. An engineer has some control over the system design variable vector (x) which influence the system attributes and costs. The engineer's goal is selecting system design variables that maximize the expected utility given the uncertainties present in exogenous variables. Product architecture is a subset of the design

variable vector with significant influence on lifecycle costs [41] and excess is proposed by this research to be another subset of this vector.

With this understanding of the various inputs for utility all that remains is its calculation. Utility is a mathematical formalism for measuring the preference for an outcome and is used to rank order design options allowing an engineer to choose which option is best given a set of preferences. Since there exists a finite supply of resources (time, money, raw materials, etc.) a designer must expend these resources in a manner which maximizes the expected utility of, or preference for, a system. In Chapter 4 the value of a gaming computer is generated by its ability to play video games at specific settings requiring a comparison of the computer's hardware with each game for a given year. The performance metric of the system is therefore the average of the fraction of games a computer can play in each year for a prescribed number of years. The system performance metric is paired with the cost of the computer and then analyzed with different customer preferences converting the value-cost pair into utility. For Chapter 6, the utility generated by the computer is related directly with computer performance as computational capacity is converted to cash via cryptocurrency mining. The preference structure is then assumed to be that the more money earned, the better. The alternative with the highest Net Present Value (NPV) is therefore optimal choice as suggested by related Value-Driven Design literature [3,42].

A principle of this framework is that a key aspect of decision-making is uncertainty. Uncertainty is the inability to exactly predict a future value in or state of the world. **Decisions made with without uncertainty (referred to as point designs** [4]**) often do not adequately characterize system performance**. One example is the original Iridium satellite network deployed to provide global cellular service. The network was a technical success but ultimately a

market failure. Lower than anticipated demand resulted in one the largest bankruptcies in the United States [27]. This example demonstrates the peril of not fully considering uncertainty during system design. Uncertainty is considered at a high-level in Chapter 6 with scenario-based exogenous variable values (discount rates, power costs, etc.).

2.6 Component-Level Change-Importance Metrics

As discussed in the introduction, reducing change propagation is a key to improving system flexibility. This section describes three techniques for identifying how important components are for change in a system: DfV, CPM, and CPI.

2.6.1 Design for Variety

DfV is an architecture approach using the four steps described in the introduction of function arrangement, assignment of function to component, mapping dependencies and using the dependency map to inform design. The goal of DfV is not creating a physical system that is easily changed, but instead creating a design that can be modified to target different market segments and changing customer preferences. DfV uses two indices for identifying which components make good reuse candidates across a family of products and which should be decoupled from the system via modularization. The indices are elicited from system experts and specified on a scale of severity.

The Coupling Index (CI) captures the strength of connections between components on a scale of [9, 6, 3, 1, 0] with 9 indicating a strong dependency. As shown in Table 2.1, a table with dependences and the weights is created and the CI is subdivided into the Coupling Index for supplying information (CI-S) and the Coupling Index for requiring information (CI-R). The CI-S is the sum of dependency weights in the column and the CI-R is the sum of dependency weights in the row.

Table 2.1 DSM Example showing calculation of coupling indices [24]

Components REQUIRING Information	Components SUPPLYING Information: Fan	Heat Sink	TEC	CI-R
Fan		Press 9 resist 3 x dim 3 z dim	Heat 3 output 1 x dim 1 z dim	**20**
Heat Sink	Pressure curve 3 x dim 3 z dim 3		Heat 3 output 3 x dim 3 z dim	**18**
TEC		Heat sink cond 3 Effective HS area 3		**6**
CI-S	**9**	**21**	**14**	**44**

The Generational Variety Index identifies how much redesign is likely required for meeting future requirements. GVI is specified on a [9, 6, 3, 1, 0] scale with 9 representing a major change incurring >50% of initial design costs and 0 representing no changes likely. These two values are used in conjunction with component redesign costs for identifying components with opportunity for reducing expected redesign costs. Plots and ranking charts are used for identifying candidate components as shown in Table 2.2. Components with low GVI and CI-R are candidates for standardization in the system platform while components with low CI-S are candidates for modularity reducing the cost of replacing the component.

Table 2.2 Results from application of DfV on a water cooler with three metrics and anticipated non-recurring engineering costs incurred by redesign [24]

Component	GVI	CI–R	CI–S	NRE $
Fascia	H	H	H	200,000
Reservoir	H	L	H	10,000
Chassis	H	H	L	40,000
Plumbing	H	L	L	1000
Heat sink	L	L	H	10,000
TEC	L	L	L	3000
Power supply	L	L	L	3000
Fan	L	L	H	2000
Insulation	L	L	L	2000

They identify four methods for reducing GVI and CI scores. These are: 1) rearranging the functional arrangement to remove troublesome engineering metric/component connections, 2) freezing a specification so that it may not be changed in future product iterations, 3) decoupling components from the system to the extent possible, and 4) increasing the "headroom" of specifications via overdesign to reduce the sensitivity of the system to change. Validation and analysis of the efficacy of the fourth method is the objective of the present research.

DfV is notable because it is an early example of both the concepts and techniques used in later changeability research. Product platforming literature refined the idea of modularization and standardization when applied to different market segments [43,44]. The notion of additional headroom they suggest is explored in detail in following chapters.

2.6.2 Change Propagation Method

The Change Propagation Method (CPM), as outlined by Clarkson et al. [23], was developed by researchers studying the effects of change on Westland Helicopters and is a foundational method for Chapter 5. CPM improves on DFV by accounting for indirect change propagation (change propagating from an initiating component through a sequence of dependencies to a component with no direct connection with the initiator).

The method begins with system experts identifying the system components and design dependencies along which a change can propagate. Dependencies are divided into direct change likelihoods (the direct probability that a change in one component will cause a change in another) and direct change impacts (the direct amount of rework required if change to one component necessitates a change in another).

Each dependency is rated on a scale from 0 to 1, with 0 being no probability of propagation, and 1 being a certainty. An example of a direct likelihood matrix is shown in Table 2.3 and is similar to a Design Structure Matrix (DSM). This matrix contains direct change likelihood probabilities elicited from system experts. The values are the elicited probabilities about whether a component modification (the k-axis) will directly cause a change to other components (on the *j* axis).

Table 2.3: Direct likelihood matrix containing the probability that one component directly changes another [23]

j \ *k*	a	b	c	d	e	f
a	-	0.3	0.3			
b	0.9	-		0.6	0.3	0.6
c	0.9		-	0.6	0.3	0.6
d	0.3	0.6	0.9	-		0.9
e			0.3	0.6	-	0.3
f	0.3	0.9	0.6	0.9	0.6	-

The combined likelihood matrix is used for calculating the total probability of change from both direct and indirect propagation. The resulting combined likelihood matrix is similar to the direct likelihood matrix except that it accounts for indirect change. The indirect likelihoods in the Forward CPM algorithm are found using an exhaustive search for all paths between two components. A description of this process is shown below using examples and figures adapted from Clarkson et al. [23]. The description is simplified with graph theory terminology. The

system is the graph and the components are the nodes. The edges connecting the nodes are the design dependencies and edge weights capture the dependency strength.

In the first step, two nodes are selected (the initiating and the target component). Each pathway between the selected nodes in the systems is enumerated using a breadth-first search starting at the initiation node at terminating at the target node without repeating any node. Figure 2.3 shows the enumeration of pathways from component **a** to component **b**. The change initiator is at the top of the tree. Each other level contains the components that are children of the nodes one step above.

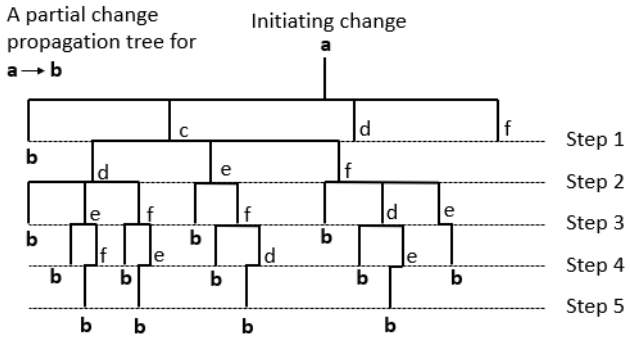


Figure 2.3: Change propagation pathway tree from initiator (a) to the target component (b) [23]

After enumerating all pathways, the resulting tree evaluated from bottom to top by a combination of AND ($\cup$) and OR ($\cap$) operations. The equations for each are:

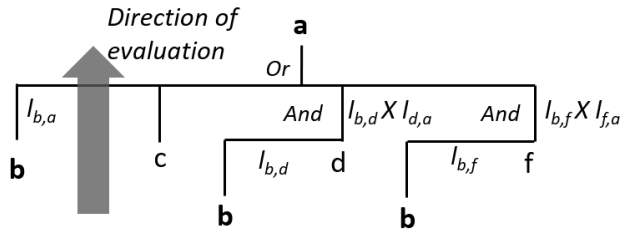
$$l_{b,u} \cup l_{b,v} = l_{b,u} * l_{b,v} \tag{2.1}$$

$$\begin{aligned} l_{b,u} \cap l_{b,v} &= l_{b,u} + l_{b,v} - (l_{b,u} * l_{b,v}) \\ &= 1 - ((1 - l_{b,u}) * (1 - l_{b,v})) \end{aligned} \tag{2.2}$$

where **l** is the direct change likelihood between component **b** and notional components **u** and **v**.

Beginning at the bottom of the tree and proceeding upward, the AND operations (Equation 2.1) collapse vertical lines and the OR operations (Equation 2.2) collapse the

horizontal lines. This process repeats for all pathways until the initiating component is reached, as shown in Figure 2.4. For a full example of this process refer to Clarkson et al. [23].



$$L_{b,a} = 1-((1-l_{b,a})*(1-l_{b,d}*l_{d,a})*(1-l_{b,f}*l_{f,a}))$$

Figure 2.4: Probability calculation using And and Or operations [23]

The resultant combined likelihood is the assessment of probability that a change will propagate from the change initiator to the target component. The process repeats for each off-diagonal cell in the matrix, yielding a combined likelihood matrix as shown in Table 2.4. The results from the calculation performed in Figure 2.4 are stored in column **a,** row **b**. CPM resolves a limitation of the Generational Variety Index by allowing change propagation across several components.

Table 2.4: Combined likelihood matrix resulting from the summation of all pathways between each pair of components [23]

L	a	b	c	d	e	f
a	-	0.6	0.1	1.0	0.1	
b	0.8	-	0.8	0.3	0.9	0.1
c	0.9	0.4	-	0.1	0.7	0.8
d		0.9	0.5	-	0.7	0.6
e	0.4		0.3	0.4	-	0.3
f	0.1	0.3	0.9	0.3	0.2	-

Koh et al. [45] refined CPM by introducing the concept of reachability. Reachability is a decay in the probability that change will propagate between components as a function of the number of steps away from the change initiator, as demonstrated in [46]. Using the formalism from the CPM example above, the equations for this adaptation are:

$$L_{k,j} = 1 - \prod_{z \in Z} [1 - (l_z * \alpha_z)] \tag{2.3}$$

$$l_z = (l_{k,k-1} * l_{k-1,k-2} * ... * l_{j+1,j}) \tag{2.4}$$

$$\alpha_z = (\alpha_{k,k-1} * \alpha_{k-1,k-2} * ... * \alpha_{j+1,j}) \tag{2.5}$$

where $L_{k,j}$ is the combined likelihood, 'j' is the change initiator, 'k' is the target component, 'z' is a single propagation pathway belonging to the set of pathways 'Z' and 'α' is the reachability. The value of α recommended by Koh et al. is 0.4 which limits the probability of change pathways longer than four steps to less than or equal to 1%. This aligns with empirical observations of change processes.

2.6.3 Change Propagation Index

CPI is a change propagation metric initially proposed Suh et al [47] and refined by Koh et al. [45]. The CPI measures the degree of propagation caused by a component when change is imposed on that component. The value ranges from [-1, 1] where -1 indicates the component absorbs change and 1 indicates the component multiplies change. The initial formulation for CPI presented in Suh is:

$$CPI_i = \sum_{j=1}^{n_{out}} \Delta E_{out,j} - \sum_{k=1}^{n_{in}} \Delta E_{in,k} \tag{2.6}$$

Where n out is set of components to which the i^{th} component is connected with an outgoing edge, n_{in} is the set of components connected to the i^{th} components with incoming edges, $\Delta E_{in,j}$ is a binary value indicating whether component j is changed by component i, and $\Delta E_{out,k}$ is binary value indicating whether component i is changed by component k. Griffen et al. [46] showed how CPI can be determined with change data from an existing system using:

$$CPI(i) = \frac{C_{\text{out}}(i) - C_{\text{in}}(i)}{C_{\text{out}}(i) + C_{\text{in}}(i)} \tag{2.7}$$

where C_{out} and C_{in} are the total number of changes in the data set that propagate from component i to any other component and to component i from any other component respectively. CPI is a metric that captures how problematic a component is with a single value. Components with large CPI values are good candidates for imbedding flexibility. Figure 2.5 shows how change propagation may occur in a system depending on the number of high CPI components in a system.

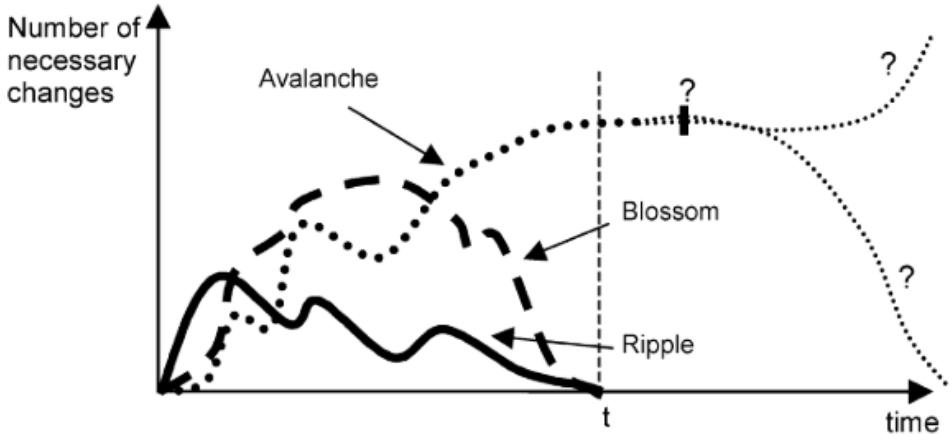


Figure 2.5: A comparison of notional change processes with increasing number of high CPI components.

A system with mostly negative CPI scores is likely to experience a decline in the number of changes over time and change propagation is largely absorbed. A blossom occurs when some change multipliers are affected but most propagation chains end in absorbers. A change avalanche occurs when "unexpected change multipliers are encountered or when change margins of known multipliers are used up." [20] If a change avalanche occurs the change may need to be abandoned or the system largely redesigned.

In this context, making a system more flexible involves finding change multipliers and reducing their propensity to propagate change. The following section details research into reducing component CPI by adding design margin (or Excess) to components to inhibit the propagation of change.

2.7 Design Margin and *Excess*

Prior flexibility research mentions the desire to "increase the headroom of specifications" [24], "control the tuning of design parameters" [48] or prevent change propagation when "change margins … are used up" [20] but these are expressed without the ontological framework and mathematical formalism required to address the desires. Design margin and excess research are introduced in this section to address this need.

Eckert et al. provides the ontological framework used in the remainder of this research [16]. A slightly modified version of defined concepts are:

- *Parameter* – a category of component or system feature (e.g. tensile strength, weight, electrical resistance)
- *Requirements* – values a parameter must reach usually imposed externally.
- *Constraint* – values a parameter must reach which are solution specific and may be intrinsic or extrinsic to the design
- *Capability* – the range of values reachable by a parameter
- *Buffer* – portion of parameter values which compensate for uncertainties in known requirements (i.e. safety margin, margin for robust design, etc.)
- *Excess* – parameter values above and beyond buffer

Figure 1.2 demonstrates these relationships graphically. Margin can exist in both "must exceed" and "must not exceed" varieties. Eckert notes that there are a variety of types of margin that may be of interest to a designer. Margin may exist on sets of parameters with a few examples being:

- MIN/MAX relationships where the margin depends on the largest or smallest parameter in the set,

- additive relationships where on the sum of the set of parameters is important (e.g. the weight of an aircraft), and
- Key Equation relationships where the parameter set is used as input to a series of relationships used to calculate a system level parameter (e.g. fuel consumption rate in a car).

Another property of margins noted by Eckert and reinforced by the author's own experience is that buffer may be converted to excess and vice versa. If the uncertainty the buffer compensates for is reduced by higher resolution analysis the unused buffer is then available for use as excess.

With the concepts related to excess defined we can now discuss prior mathematical models of excess. Initial *excess* research focuses on *excess* at the system level via parameter sets. Tackett el al. [49] developed a mathematical relationship based on Hooke's law between *excess* and system's evolvability defined as "the potential . . . for a system to evolve from one configuration to another . . . to meet specific new system objectives." Tackett stated that e*xcess* is consumed by changes (e.g. using *excess* electric energy to power a new electromagnetic air launch system). Once *excess* is consumed the system is no longer evolvable likely resulting in obsolescence. Tackett stated that the relationship for a system's capacity for change (C) and excess (X) are related by a gain term (g_x) that accounts for the value of the specific type excess as shown in Equation 2.8.

$$C = g_x X \tag{2.8}$$

The total evolvability of the system (E) is then the integral of the capacity across all types of excess as shown in Equation 2.9 where x_u and x_l are the lower and upper bounds of useful excess.

$$E = \int_{x_l}^{x_u} g_x X dX \tag{2.9}$$

Allen et al. [50] introduce the notion of the usability of *excess* based on its "type, location, and form." Different forms of excess flows were identified as shown in Table 2.5.

Table 2.5: Types of excess capability and their associated parameters [50]

Type	Parameters
Volume, space	Length, width, height
Electrical power	Voltage, current, or amplitude, frequency and phase
Kinetic translation energy	Mass, velocity
Kinetic rotational energy	Moment of inertia, angular velocity
Potential energy	Mass, distance, length, or force, length, or Watts, or Joules
Pressure	Force, area
Torque	Force, moment arm length
Information, data transfer	bps, time, frequency
Electromechanical	Current, force, field strength
Chemical	Enthalpy of formation, reactivity, *p*H, net charge
Thermal	Specific heat, conductivity, density, enthalpy
Sound	Amplitude, frequency
Nuclear	Decay rate, radioactivity, density
Structural weight	Density, volume
Buoyant weight capacity	Volume, displacement
Volume flow	Volume, velocity

Allen explored the notion that e*xcess* is only valuable if "the *excess* capability can be used to fulfill specific future requirements." This definition requires that *excess* be evaluated based on anticipated future requirements and introduced a usability factor (q) based on that evaluation as shown in Equation 2.10 where x is they type of excess, x_a is the excess available in the system and x_r is the quantity of excess required for future needs.

$$qx = \begin{cases} \dfrac{x_{\text{avail}}}{x_r}, & \text{if } x_a < x_r \\ 1, & \text{if } x_a \geq x_r \end{cases} \tag{2.10}$$

Watson et al. [51] build on this model of *excess* applying it to a simplified model of a military ground vehicle and optimize the quantity of *excess* available. The optimization involved the value of meeting specific hypothetical requirements and the costs of the *excess*.

Recent work by Cansler et al. [52] and White and Ferguson [53] apply the concept of *excess* at the component level. Each paper breaks simple electromechanical systems into components and identifies flows, Table 2.5, between them. These efforts begin to identify how *excess* in individual components can improve a system's changeability. Figure 2.6 shows a heating element from a heat gun as an example showing flows into and out of the component.

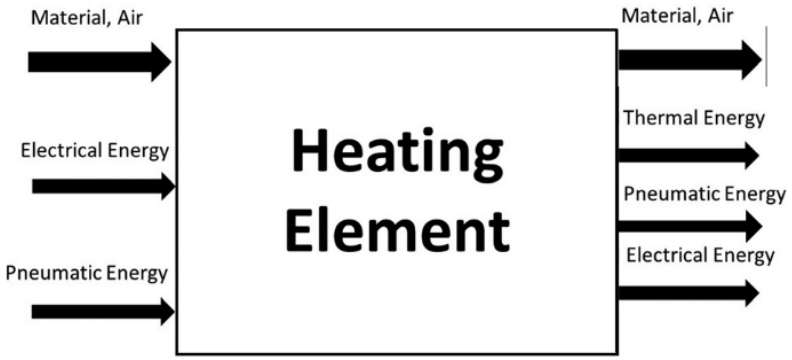


Figure 2.6: A heating element example of a component flow model [52]

The flows for each component were identified and the key equation relationships for specific system level parameters mapped. The system level parameters were then modified to stress test the system to identify which component level parameters limit changes in system level parameters.

2.8 Chapter Summary

Prior excess research has begun the exploration of using excess for increasing system flexibility, but the concept is still relatively new. Excess research is currently positioned in an analogous way to early modularity research with some supporting evidence and approaches developed, but more work required. The following chapters provide further evidence of the value for including excess (Chapter 3 and Chapter 4) and then develop new approaches for including, measuring, and evaluating component excess (Chapter 5 and Chapter 6).

Chapter 3: Qualitative Excess Assessment of Two Historical Military Aircraft

3.1 Introduction

The purpose of this chapter is to address research question 1: Does qualitative evidence of the presence or absence of *excess* in the initial design of a system support the hypothesis that *excess* influences system lifecycle performance? Previous *excess* research focuses on either non-complex systems or applies the proposed methodology to a complex system in the context of hypothetical changes. Few other works have examined the evolutionary paths (the sequence of system modifications and *excess* consumption) that existing complex systems have taken in response to changes in their context.

Several candidate systems were screened for study including commercial aircraft, military aircraft, space systems, nuclear power plants, and large infrastructure systems. The assessment of candidate systems focused on systems with characteristics of complex systems [2], the availability of sufficient public information to perform a qualitative analysis, and the noteworthiness of how the system's architecture and parameter choices impacted its flexibility. Most screened systems were found to have insufficient public information available to properly study their evolutionary trajectories. The search was therefore limited to military aircraft due to the requirement for availability of historic records. These systems were found to have government documents for design details and a plethora of aviation enthusiast's accounts providing context for the system's lifecycle.

Ultimately two systems were selected for the study due to the third criteria requiring noteworthy interactions between initial system design and lifecycle performance. The two selected systems were the B-52 Stratofortress and the F/A-18 Hornet. The B-52 was selected due to its lengthy operational history and proven ability for adapting to technological challenges for

65 years with another 20 planned. The F/A-18 was selected because it was unable to evolve to meet new operational requirements. A public debate between the U.S. Navy and the U.S. GAO provides public record of design rationale that ultimately resulted in a redesign as the Super Hornet leaving only 15% commonality with the original airframe [54].

3.2 Method for Case Study

Research question 1 is addressed via a retrospective lifecycle analysis of the two selected systems. These accounts are produced in two steps. First, aviation historical accounts were compiled for each aircraft. These are narrative accounts of the lifecycles for each aircraft which provide context with varying degrees of detail. Once the context and general overview for each lifecycle is compiled a search of government records was conducted to fill in modification details with contemporaneous rationale for specific decisions.

The accounts of each aircraft are documented sequentially. First is an analysis of the initial design context and the influence that context had on design requirements. Second is an analysis of the different context changes imposed on the system throughout its life and the modifications made to adapt the system for those context changes. This includes a qualitative analysis of *excess* types consumed in support of those modifications. The final section for each aircraft is a system specific discussion.

The B-52 section concludes with a discussion of how the initial design context and requirements supported the development of a system with sufficient *excess* to be successfully adapted to context changes throughout its very long life.

The F/A-18 section proceeds with an analysis of the circumstances that lead to the required redesign. The focus is on the three deficiencies that were symptomatic of insufficient system *excess*: a range/payload shortfall, the inability to support internal system growth, and the

degradation in payload bring back. The F/A-18 section then concludes with a discussion of how the interplay of these deficiencies resulted in the system redesign.

3.3 The B-52 Stratofortress

3.3.1 Historical Context

At the close of World War 2 and the opening of the Cold War the United States found itself in need of a bomber that could deliver an atomic bomb to targets far from United States Air Force (USAF) bases. Initially only propeller aircraft like the B-36 could provide the desired range, but they were significantly slower than newer jet bombers like the B-47. Jet bombers were capable of faster speeds, but suffered from reduced payload capacity and range [55]. Between 1945 and 1948 the Air Force released increasingly challenging specifications for the new bomber and Boeing iterated through many different configurations. Finally, the USAF settled on requirements for a bomber with a range of 8,000 miles and a minimum cruising speed of 550 mph that was capable of delivering an atomic bomb while flying above the effective anti-aircraft gun range.

The B-52 architecture has been leveraged in 8 different versions, designated with letters A-H. Table 3.1 indicates how the design changed between generations by considering maximum take-off weight and fuel capacity [56–58]. This list does not include the A model as it underwent limited production and was primarily used for testing and evaluation. The final “H” model was fitted with new turbofan engines, extending the range to 4,825 miles with capacity for 10,000lb of ordinance. The discussion in this paper primarily focuses on the G and H models.

Table 3.1: A Comparison of Select Attributes for B-52 models [56–58]

Model	Max Take-off Weight (1000's lb)	Fuel Capacity (gal)	Original Empty Weight (klb)	Radius with 10k lb bomb load (mi)	Max Military Load (1000's lb)
B	420	37,750	164	3,590	63
C	450	41,700	178	3,475	64
D	450	41,550	178	3,305	65
E	450	41,550	175	3,500	65
F	450	41,550	174	3,650	65
G	488	47,975	168	4,100	105
H	488	47,975	173	4,825	105

The context in which the B-52 was designed had unique requirements that resulted in a design with significant *excess* in the bomber's size, weight, and range. The primary requirements were an 8,000-mile range, a minimum cruising speed of 550 mph, and a capacity for 10,000lb of ordinance. Table 3.2 shows a comparison of the max takeoff weight, fuel capacity, radius, and maximum military load between the three US strategic bombers currently operational. The B-52 was not subject to requirements associated with traveling at supersonic speeds (like the B-1B) or requirements associated with minimizing radar observability (like the B-2) and was designed prior to the realization of in-air refueling. Consequently, it is larger, heavier, and possess a superior range compared with the other two strategic bombers. In the following sections argue that it is precisely these parameters that enabled the B-52 to adapt to the many new requirements placed on it over its lengthy in-service period.

Table 3.2: A comparison select parameters between operational US strategic bombers showing parameter *excess* with respect to the requirements of newer aircraft [59–61]

Aircraft	Max takeoff Weight (klb)	Fuel Capacity (klb)	Unrefueled Range (mi)	Max Military Load (klb)	Unit Cost ($M 1998)
B-52H	488	319	9,650	105	60
B-1B	477	265	7,456	75	283
B-2	337	167	6,905	40	1,157

3.3.2 Lifecycle Analysis

Originally designed as a nuclear bomber before the advent of effective surface-to-air missiles (SAMs), the B-52 faced many challenges to remaining an effective military system. Adaptation to challenges often required physical modifications and operational profile changes. The adaptations described in the following sections are grouped by change driver. These include: the development of accurate SAM's, the need to deliver conventional and precision guided payloads, and the integration of modern electronics.

3.3.2.1 Surface to Air Missiles (SAMs)

The original concept of operations for the B-52 used altitude and speed to shield from anti-aircraft batteries. The first operational challenge was the development of SAMs, which posed a threat to the fulfillment of its mission to deliver free-fall nuclear weapons [62]. Changes to the bomber and its mission included three primary adaptations [57]:

- change of mission flight path from high to low altitude below enemy radar (300-500 ft.)
- development of sub-systems capable of defeating tracking systems on adversary weapons
- development of stand-off weapons alleviating the need to penetrate as deeply into hostile airspace

Each adaptation required supporting modifications to the B-52. Each is discussed in more detail below.

3.3.2.2 Change in Operational Altitude

The change in operating altitude required many apparent changes. Flying at low altitude requires the addition of subsystems to avoid ground collisions and modified targeting systems. Most modifications were incorporated with the "Mod 1000" upgrade where the aircraft were equipped to carry "improved bombing-navigation systems, Doppler radar, terrain avoidance

radar, and low-altitude altimeters." [63] Additional improvements were made in the intervening years to add multiple sensor and computer modifications enhancing terrain following capability including the Electro-optical viewing system and the "Jolly Well" upgrade to the ordinance and navigation system [57]. These modifications added weight, used internal volume, and required exterior modifications deteriorating aerodynamic efficiency.

The unforeseen and poorly understood phenomenon of fatigue failure accompanied the transition to lower altitudes [58]. Increased turbulence at lower altitudes induced fatigue stresses on the airframe and resulted in two separate incidences of the vertical stabilizer failing mid-flight [56] and the appearance of wing cracking [62]. A modification program called "Hi-Stress" was implemented to provide structural modifications to support low-level flight. Modifications included: "strengthening the fuselage bulkheads, aileron bay area, boost pump access panels and wing foot splice plate, upper and lower wing panels, upper wing surface probe access doors, and the bottom portion of the fuselage bulkhead." [63] Later the "Pacer Plank" and ECP1050 programs further strengthened bomber airframes [57]. These modifications added additional weight to the aircraft.

3.3.2.3 Survivability Enhancements

The second adaptation increasing survivability was the development of systems aimed at defeating the tracking system on enemy missiles. These modifications included decoy missiles, enhanced ECM systems, and other countermeasures.

The Quail missile was an air launched decoy designed to present a large radar cross-section and intense infrared signature [64]. The B-52 could to carry up to 8 but the typical load-out was for two. These weapons were carried on newly installed external pylons. The Quail

missiles and the external pylons added weight and another penalty to the aircraft's aerodynamic efficiency.

The B-52 ECM systems were constantly upgraded as part of the race between enemy targeting systems and bomber defenses. Programs that enhanced ECM capability included "Mod 1000", "Rivet Ace", and "Rivet Rambler" which added a host of radars, false-target generators, jamming equipment and flare/chaff dispensers designed to protect the B-52 [57]. These were all fitted onto or within the space allowed by the airframe.

3.3.2.4 Standoff Weapons

As anti-aircraft weaponry improved it became clear that sending the B-52 into hostile airspace was an unacceptable risk. The development of better guidance technology for missiles allowed for the possibility that a B-52 may never have to enter threatened space. Instead the bomber would carry weapons that would be deployed at a distance as to not endanger the bomber.

The first program was the development of the "Hound Dog" nuclear missiles that were designed to penetrate Russian defenses during the Cold War. Further improvement in missile technology led to the integration of the Short Range Attack Missile (SRAM) and the Air Launched Cruise Missile (ALCM) [63]. Each of these required supporting systems that were installed in the ECP2126 program that involved "the addition of modified wing pylons and launch gear as well as weapons bay rotary launchers and associated avionics equipment" [57] and the ALCM integration which included "new digitized offensive avionics systems" which allowed the B-52 to carry 20 AGM-86 ALCMs [63].

3.3.2.5 Conventional Weapons Modifications

The B-52 was originally designed in an era during which war planners believed that strategic bombers would primarily use nuclear weapons during conflicts. As history shows this was not to be the case and delivery of conventional weapons over to countries far from U.S. air bases was required for conflicts like Vietnam. The development of in-flight refueling changed the limiting attribute from range with a certain payload to how much weight and volume could be accommodated by the airframe.

The B-52 has been repeatedly modified to carry heavier and larger conventional weapons. The initial modifications involved the introduction of an external pylon on which weapons could be carried. This was first used to carry the Hound Dog nuclear missile, but was later adapted to carry SRAMs and ALCMs, and with the introduction of the Heavy Stores Adapter Beams could carry additional weapons including weapons too long or large to fit the original I-beam [62]. The D model, primarily used in Vietnam, had its bomb bay modified to carry additional bombs internally on high density racks nicknamed the “Big Belly” modification increasing carrying capacity “for a maximum bomb load of about 60,000 pounds – 22,000 pounds more than the B-52F” [58].

The capability to reduce collateral damage while expending less ordinance to destroy a target led to the introduction of smart weapons. These weapons require that targeting information be conveyed to the weapon which meant both the targeting information had to be generated via positional system and communicated to the weapon. The conventional enhancement program that included the “Rapid Eight” effort added the necessary enhancements. These included: a GPS navigation receiver, VHF/UHF radio with VHF/UHF and satellite communications capabilities, and the MIL-STD-1760 databus for weapon on its external pylons [62]. These enhancements

allowed the B-52 to carry a variety of weapons including the Have Nap, joint stand-off weapon (JSOW), and joint air to surface stand-off missiles (JASSM) in addition to JDAM guided bombs [62]. Additional enhancements are planned to allow these weapons to be carried internally [65]. With these enhancements, the large internal weapons bay, and the external pylons the B-52 can carry a larger variety of weapons than any other bomber in the US Air Force. A visual comparison of the available bomb bays and the variety of ordinance supported by each of the UASF's strategic bombers is shown in Figure 3.1.

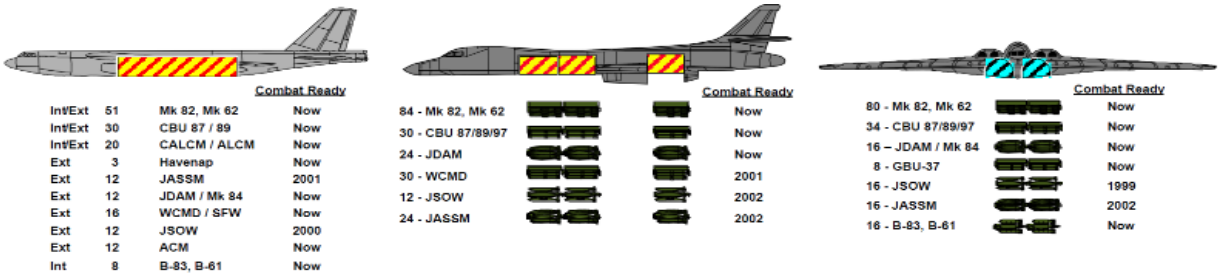


Figure 3.1: Ordinance comparison for modern US Bombers [66]

3.3.2.6 Modern Electronics Integration

Following the conclusion of the Vietnam conflict the B-52 has experienced modernization upgrades primarily made possible by the miniaturization of electronics. The development of enhanced computing allows constant communication with resources on the ground and satellites overhead. The B-52 is currently undergoing a computational overhaul to prepare it for operation until 2040. The CONECT program is responsible for integrating the B-52 with "Air Force communication networks and platforms… to receive mission data in flight and retarget weapons" [67]. According to the US Air Force the CONECT program would involve "upgrading the B-52 fleet with tactical datalink and voice communications capability along with improved threat and situational awareness to support participation in network centric operations" [65]. This would allow the B-52 to become and integrated part of the battlefield by allowing information sharing while in-route or over target.

3.3.3 B-52 Summary

The B-52's mission has change dramatically from the initial mission to fly higher and faster than enemy air defenses to deliver nuclear weapons in the 1950's to the role it played in Afghanistan which involve loitering over the battlefield to deliver smart weapons for close air support [68]. It has successfully adapted to: new defensive requirements necessitated by the introduction of SAMs, operation at a more challenging altitude, the introduction of ever more advanced weaponry, and the modern necessity of being electronically connected to an integrated digital battlefield. The 2014 Congressional Report summarizes the bomber saying, "The B-52's strengths lie in its diverse capabilities, precision, large payload, and long range; however, if these capabilities remain static, mission effectiveness is likely to erode in the face of 21st century … threats" [69]. The adaptations and associated modifications were enabled by an airframe with significant *excess* weight, range, and volume suggesting that ample *excess* supports flexibility.

3.4 The F/A-18 Hornet

3.4.1 Historical Context

The series of events resulting in a need for lightweight fighter aircraft can be traced to just after the Korean War. Planners felt that future aerial engagements would be fought beyond visual range with new missile technology. This thinking emphasized the need for high-speed interceptors and deemphasized the need for maneuverable air superiority aircraft. However, the Vietnam conflict demonstrated flaws in planner's assumptions with repeated losses to inferior North Vietnamese fighters. These "galvanized sentiment in the Air Force for a new air-superiority fighter." [70]

As a result, the USAF and Navy pursued the FX programs that led to the development of the F-14 and F-15 fighters that were optimized for air superiority. These aircraft were designed

to dominate the skies but were also expensive. The high cost meant they could only be produced in limited numbers. A strategy for using a mix of cheaper light weight fighters with their more expensive counterparts was developed called the “high-low” mix. This strategy ensured sufficient aircraft would be available for future conflicts with existing defense budgets [71]. A request for proposals for the light-weight aircraft was released in 1972 called the Lightweight Fighter Competition (LFC).

The final fly-off for the LFC had two competitors: the General Dynamics YF-16 and the Northrop YF-17 Cobra. The outcome of the fly-off was selection of the F-16 by the Air Force due to its slightly higher speed and commonality with the F-15 engine [72]. The Navy, however, was unhappy with the decision as they felt it would be too costly to adapt the F-16 for carrier operation. Instead the Navy funded development of the YF-17 into what would become the F/A-18 Hornet.

The F/A-18 was originally intended to have two variants: one model optimized for the attack role and the other for the fighter role. Sufficient advances in radar design, stores management, and multifunction displays occurred allowing the two models to be merged into a single aircraft [72]. It was equipped with the first all-digital fly-by-wire system and demonstrated high level of reliability and maintainability [73]. The F/A-18 was a versatile system that could fulfill the roles of the F-4 and A-7, both of which it replaced, and was the first modern aircraft with the dual classification of Attack and Fighter.

A consequence of this context was ambitious requirements. The F/A-18 was expected to be a lightweight, low-cost, multi-role, and carrier borne aircraft. The resulting design was therefore subject to many different constraints that Northrop had difficulty satisfying. Several deficiencies were highlighted by a congressional report including inability to meet the specified

range and weight requirements [73]. This implies that the F/A-18's design had little or no *excess* available for future modifications that would adversely impact either of those parameters. This was highlighted in the report stating that "… the potential for additional unacceptable weight growth does exist." Additionally, the costs for the program reference in the report were expected to be $24 billion, nearly twice initial estimate of $13 billion, further constraining Northrop's ability to add *excess* to the aircraft's design.

3.4.2 Lifecycle Analysis

Modifications and enhancements were made to the F/A-18 in the mid 1980's with the new variants given designated C/D variants. Improvements included: "a revised … ejection seat, improved XN-6 mission computers, upgraded stores management set, an upgraded armament bus (MIL-STD-1553B and -1760), a flight incident recorder and monitoring set (FIRAMS)" [72], a new ECM system (ALQ-165), and a new warning radar [54].

A second round of improvements occurred 1988 which give the Hornet the ability to operate effectively at night giving the modified aircraft the moniker "Night Attack". This package included: GEC-Marconi AXS-9 night vision goggles, two new 5x5 color multi-function display screens, and a color digital moving map display. An infrared pod was added to the right fuselage station which provided a Forward Looking Infrared (FLIR) overlay on the heads-up display. The canopies of the fighters were tinted with gold to help deflect radar energy to minimize radar cross-section. A new software package was also included which combined the sensor information received into an integrated picture of what was occurring outside the Hornet reducing pilot workload and enhancing targeting [54].

Other improvements were made following the night attack modifications. The APG-65 radar was replaced with the APG-73 which provided significantly better performance. A "high-

resolution synthetic aperture radar" mode was added providing enhanced ground mapping and allowed for "autonomous targeting for the AGM-154 Joint Stand-Off Weapon (JSOW) and the GBU-32 Joint Direct Attack Munition (JDAM)" [72]. A GPS receiver and an enhanced IFF transponder were added along with the capability to carry AIM-120 AMRAAM missiles were all incorporated in the mid 1990's [54].

In the early 1988 Boeing and the Navy recognized that the current airframe would be pushed to its limit. Boeing released a study called "Hornet 2000" in which "… seven configurations were evaluated on a variety of factors, including carrier suitability, strike and fighter missions, maneuverability, systems, survivability, growth, effectiveness and costs" [54]. From these variants, a new aircraft was designed that combined the best aspects from each design while maintaining affordability.

Table 3.3: Comparison of Select Attributes of F/A-18 Models[54,73,74]

	YF-17	F/A-18 A/B	F/A-18 C/D	F/A-18 E/F
Empty Weight (lb)	17,000	21,830	24,372	30,564
Internal Fuel (lb)		10,860	10,860	14,700
External Fuel (lb)		6,700	6,700	9,800
Wing Area (ft^2)	350	400	400	500
First Flight	June 9, 1974	November 18, 1978	September 3, 1987	December 1, 1995

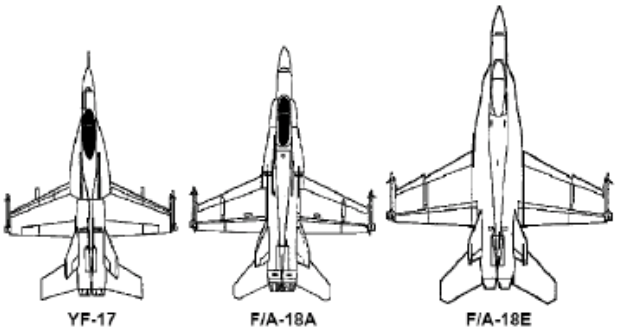


Figure 3.2: Visual Comparison of F/A-18 Models [74]

The designation of E/F was given to this plane even though it was essentially a new design. The F/A-18 E/F had approximately 10% commonality with the F/A-18A. It was 25%

larger with a 1/3 fuel capacity boost, larger control surfaces, and a 42% reduction in parts [75]. Table 3.3 and Figure 3.2 provide a comparison of the statistics and a visual comparison of the fighter in its different phases. The E/F models finally resolved issues which had plagued the program from its inception as discussed in the next section.

3.4.3 Symptoms of Insufficient *Excess*

The original F/A-18 was officially operational for 5 years before the need for a new design was recognized. The effort was initiated with the "Hornet 2000" study in 1988 and continued into the mid 1990's. The decision to substantially redesign the Hornet faced a great deal of criticism from the GAO which argued that the alternative solution of further modifying the C/D models was cheaper and sufficiently effective to allow time for the development of the next generation warplane.

There were several deficiencies and new external requirements which, individually, could have been rectified or accepted, but in concert they provided sufficient justification to require redesign despite GAO concerns. These shortcomings fell under the categories of: Range/Payload, support for further internal systems growth, and payload recovery. A discussion of these shortcomings and the design changes made for the F/A-18 E/F variants follows showing how each deficiency was addressed by adding *excess*.

3.4.3.1 Range/Payload

One of the concerns that contributed to the YF-17s loss at the original LWF fly offs was the aircraft's range. The redesigned F/A-18 continued to suffer from a range deficiency during development which was cited in multiple GAO reports [73]. The Hornet faced its greatest range deficit in its attack configuration and the production of aircraft needed to fulfill attack role was nearly canceled. Support from the naval community was sufficient to continue the program, but

additional modifications were made to help address the issue during pre-production. These modifications included adjusting the angle of the leading-edge flaps and filling in part of the boundary layer air discharge slots which were found to increased drag. These changes helped to increase range, but also led to a change in air flow such that the vertical stabilizers would eventually experience fatigue issues [54].

Ultimately the Navy accepted the range deficiency in the production model. The rational was that the range was short of what was desired but still acceptable and that aerial refueling would provide compensation for missions which required longer ranges.

Table 3.3 shows that the fuel capacity between the A and C model remained unchanged. The Hornet 2000 study examined the issue of increasing the range of the Hornet. The simplest way to increase range was to add more fuel. This fuel could be added by either increasing the internal storage space allotted for fuel or by increasing the volume of the external tanks carried by the Hornet.

The GAO proposed using larger 480-gallon drop tanks instead of the traditional 330-gallon tanks on the C/D models to enhance range without developing a new aircraft. The Navy responded that this idea had been studied and that the stress on the aircraft when being catapulted off the deck was “above design limit load” [76]. In order to use the tanks, the airframe of the Hornet would have to be strengthened which would involve added weight and cost.

The redesigned Super Hornet included both larger internal fuel capacity and increased volume external tanks to resolve the deficiency. A 2.3 ft. fuselage plug added some the extra internal space for fuel [72]. The external tanks were also increased in size from 330-gallons to 480-gallons by using a new filament winding technology with a toughened resin system. The new technology allowed the tank to only increase diameter 3.1 inches and provide the same

empty weight. Figures 3.3 and 3.4 visually depict the differences in the locations in which fuel is carried in each variant.

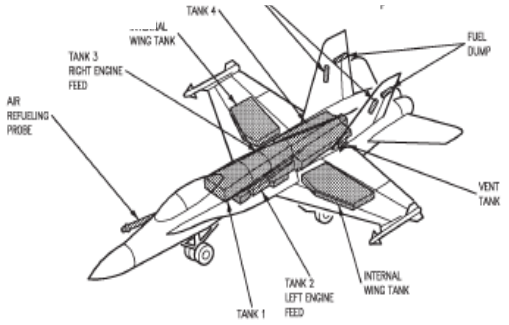


Figure 3.3 F/A-18 C Fuel Storage [77]

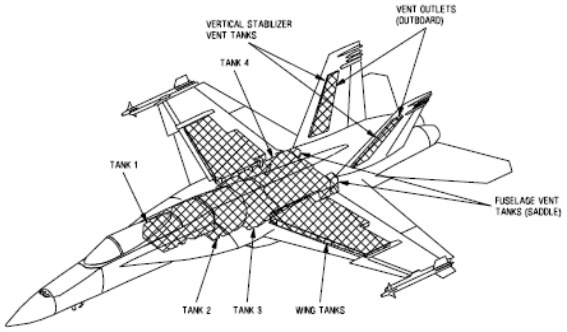


Figure 3.4 F/A-18 E Fuel Storage [78]

The combat radius in the fighter profile increased by 54 nautical miles, and the attack profile range increased by 75 nautical miles with the increased fuel and aerodynamic enhancements [79]. With these additions, the F/A-18 E/F was able to meet the design goals for range/payload specified by the Navy.

3.4.3.2 Internal System's Growth

As the F/A-18 evolved new systems were added internally to expand the capabilities of the system. This growth led to the prediction in 1992 that by 1996 additional upgrades would have insufficient internal space available. Additionally, the Navy claimed that there would be insufficient power and cooling for new systems in the aircraft [54]. The GAO report argued that miniaturization of existing systems would provide sufficient room. A detailed breakdown of projected weight/volume savings is shown in Table 3.4.

Table 3.4: Proposed F/A-18 modifications with associated weight and volume changes showing miniaturization required to support desired modifications [76]

Effect of Replacing Avionics Systems on the F/A-18 Hornet				
Equipment	**Old system**	**Replacement system**	**Weight (pounds)**	**Volume (cubic feet)**
Radar	APG-65	APG-73	–12.0	–0.90
Communication receiver/ transmitter	ARC-182 (2)	ARC-210 (2)	+5.6	+0.12
Chaff countermeasures set	AN/ALE-39	AN/ALE-47	+22.7	–0.14
Missile command launch computer	AWG-25	AWG-25 MOD Downsized HARM	–11.0	+0.01
Weapon station management system	SMS	SMS (upgrade)	–71.9	–1.20
Countermeasures receiving set	ALR-67(V)2	ALR-67(V)3	–8.4	–0.30
Global positioning system	MAGR	EGI Combined GPS/INS	–38.6	–0.63
Inertial navigation system	ASN-139A			
Total			**–114.0**	**–3.0**

The GAO also pointed to space available in one of the LEXs and extra space that would be available if the fighter were to switch to caseless ammunition for its gun. The DOD argued that space in the LEX and gun bay experienced higher levels of vibration than avionics could with withstand. They also stated their belief that miniaturization of the systems listed above had and would continue to add significant cost to the development of each system. The amount of space added by miniaturizing planned electronic systems would be insufficient to support long term system growth.

The F/A-18E/F was designed from the outset to include space to be filled by the addition or modification of future systems. Boeing reserved 17 cubic feet of internal space along with *excess* electrical power and cooling capacity [72] to allow the aircraft to evolve successfully without the need for concern about available space or expensive miniaturization.

3.4.3.3 Payload Recovery

Payload recovery is the weight of unused stores, fuel, and external equipment the aircraft can return safely to the carrier. Carrier landings are stressful to the airframe and landing gear due to the sudden acceleration and limits are imposed to minimize the risk of damage.

A Navy official when interviewed about the issue summarized the situation saying that [54]:

"… the Hornet's bring-back was starting to erode. This occurred, in turn, at the same time we were seeing an increased emphasis on 'smart' weapons as well as increased cost of these weapons. Our options were: 1) land with less fuel, which presented one set of dangers; 2) land with less weapons, which meant dropping unused weapons before landing; or 3) not carrying as much fuel or weapons. None were attractive and all hampered the mission."

The Navy projected that the weight growth of the F/A-18 combined with the weight growth of new weapons had the potential to cause mission planning problems. The original F/A-18C had a payload return capacity of 6,300lbs. Projections from 1992 showed that by 1995 this would be reduced to 5,785lbs by the weight of additional systems.

The weapon systems were also transitioning to precision guided variants. Precision guided weapons are generally constructed by taking a dumb bomb and adding control surfaces and guidance hardware. These modifications add weight to each class of weapon carried. Table 3.5 shows the weight difference between the variants that the F/A-18 was qualified to carry.

Table 3.5: Weight Increase of Precision Munitions [80]

Guided Bomb	Weight (lb)	Equivalent Dumb Bomb	Weight (lb)	Weight Difference (lb)
GBU-10	2,153	MK-84	2,031	122
GBU-12	619	MK-82	514	105
GBU-16	1,131	MK-83	1,005	126

The GAO stated that the Hornet 2000 study suggested the recovery weight for the F/A-18 could be further increased by strengthening the landing gear such that payload recovery could be increased by an extra 3000lbs. The engineering reality was that strengthening the landing gear and airframe would add weight to the aircraft. This weight would further increase the air-speed of the approach and would require a larger wing area to compensate [72] since the Hornet's landing speed started above the original design specification. This spiral would essentially lead to a larger aircraft which is what the Super Hornet already represented.

The Navy did increase the allowable bring-back weight of the original Hornet by 1,000 lbs to allow additional payload recovery through the use of "minor flight control software and procedural changes" [72] but continued growth made this a temporary solution.

The Super Hornet was designed to allow for significantly greater bring back capability. The maximum carrier landing weight increased to 42,900lbs which despite the heavier airframe allowed almost 200% greater actual payload return [72].

3.4.4 F/A-18 Summary

The initial F/A-18 design was a highly versatile and relatively low-cost system. It was capable of operating from an aircraft carrier while performing both fighter and attack roles in a platform that cost $43M per aircraft compared to the E/F model which cost $95M [79]. A consequence of versatility at low cost is the system had difficulty meeting the initial design requirements specified by the Navy and subsequent modifications worsened the deficiencies.

Evidence suggests that insufficient *excess* was included in the original design to support the system evolvability. The symptoms of insufficient *excess* were: range/payload insufficiency, inadequate internal volume/power for additional subsystems, and eroded payload recovery. The system was substantially redesigned as the F/A-18E variant which added the size necessary to both fulfill initial design objectives and provide *excess* for future evolutions.

3.5 Discussion

An examination of the evolutionary trajectory for the B-52 and F/A-18 reveals two dissimilar paths. The B-52 experienced a lengthy in-service period relative to other strategic bombers while the initial F/A-18 had a relatively short operational period before the need for redesign was recognized. The preceding system analysis enables discussion of what may be learned about *excess*. The key insights about system design and operation are listed as follows.

1) Ease of change does not ensure system longevity.

The F/A-18A/B was designed with many characteristics making modifications easier to implement. Design features like a digital architecture with a multiplex bus allowed sensor and weapons systems upgrades to be incorporated with greater ease [81]. Despite these considerations, the original F/A-18 airframe had insufficient design *excess* in critical areas that did not provide for future system growth.

Using the F/A-18 as an example a depiction is shown in Figure 3.5. In the change absorption regime modifications incur minimal change propagation. The modification cost is driven by the design, development, and modification of hardware related to the desired modification. The more change propagates the higher this constant is. The modifications for the upgrade to the C/D variant fall within this portion of the graph.

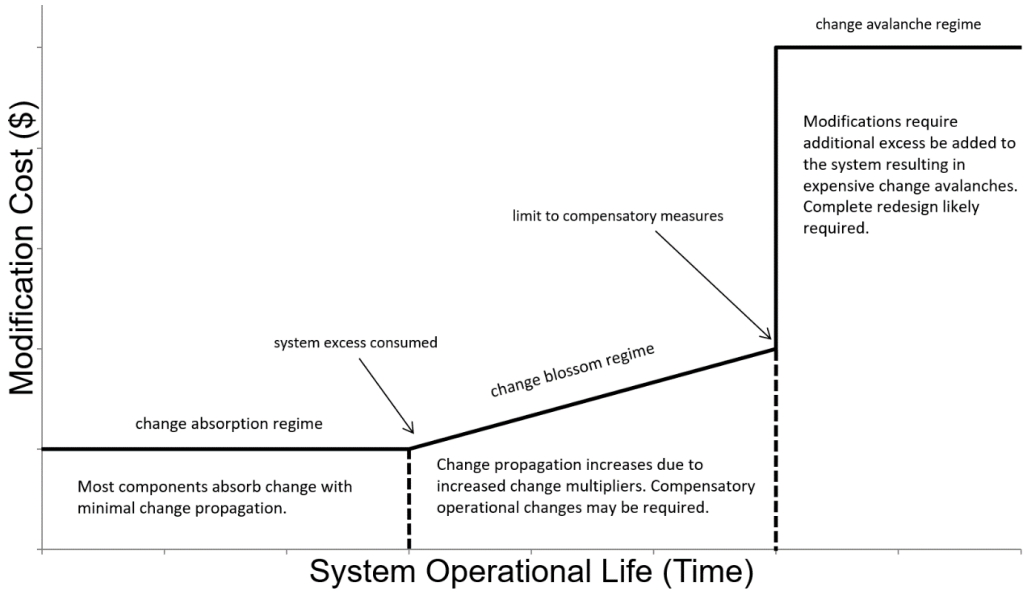


Figure 3.5: Notional chart showing how the consumption of *excess* increases costs of modifications until the system must be redesigned

Once weight growth from system additions reached a threshold, compensatory measures were required for further modifications. For example, pilots dumping unused ordinance into the ocean before landing. Additionally, change propagation became more prevalent. In order to

provide more internal volume and decrease weight miniaturization of existing electronics was required. These each incur additional costs shown in the change blossom regime on the chart.

Finally, the design was pushed to its limit and a substantial redesign was required incurring a step change in cost to add further system resources for future adaptation.

2) Change drivers for a system are stochastic in time and severity but *excess* can support adaptations for increasing system lifecycle value.

The performance shortcomings resulting in the F/A-18 redesign included insufficient internal space, insufficient *excess* carrier landing speed, and limited range/payload. Each of these performance issues stemmed from a lack of *excess* incorporated into the original design. It is perhaps true that system growth occurred more quickly with the F/A-18 than with other aircraft, but weight growth is common [82] and provision for it during initial design could have prevented the need for redesign.

In contrast the B-52 included ample *excess* sustaining system flexibility throughout its life. These include a large capacity for weight growth, aerodynamic efficiency allowing for acceptable performance after degradation of additional systems, landing gear placement allowed for carriage of large weapons, structural modifications allowing for durable airframe, high degrees of redundancy, and sufficient internal room for system growth [25]. These features combined ensured that each challenge to the B-52's system capabilities was at least partially mitigated by an appropriate adaptation.

3) Specific types of excess have some potential for fungibility.

The performance degradation experienced by the F/A-18 required *excess* to overcome, but some types of *excess* could offset the need for other types. For example, the Hornet was plagued by a shortfall of internal space needed both for extra fuel and internal systems growth. If

the airframe and landing gear had been more robust it is possible that larger external fuel tanks could have been used offsetting the need for internal fuel volume growth. A larger wing area would create additional lift at low speeds thereby decreasing the take-off and landing airspeed and reducing the forces experienced by the aircraft. Lower transient forces allow larger external fuel tanks again offsetting the need for additional internal space for fuel.

This phenomenon is more pronounced in the B-52 for two particular adaptations. The first is the collection of modifications necessary for surviving the challenge posed by SAMs. The change in operational altitude forced designers to reinforce the airframe and skin of the bomber so it could endure the more turbulent low altitude environment. If the B-52 had been designed with less *excess* to a weight constraint it may not have been possible to add the additional weight necessary for reinforcement. Weight *excess* was thereby traded for structural strength.

The second noteworthy modification was the addition of pylons and pylon extensions to carry ordinance externally. In essence, these modifications used structural strength and aerodynamic efficiency *excess* to offset the need for additional internal volume. Insufficient *excess* for supporting externally carried ordinance would have reduced the value of the system by limiting the dimensions and quantity of ordinance the aircraft was able to carry.

4) The magnitude of modification excess consumption should be considered relative to the total magnitude of the associated system parameter.

While the B-52 has experienced significantly more challenges over its longer life, the two systems overcame similar challenges posed by technological advances. Each aircraft was adapted to support precision guided weaponry, night attack capability, and the integration of equipment necessary for a data-centric modern battlefield.

The modifications performed on each system required consumption of *excess* of a similar magnitude. GPS equipment was added to guide precision weaponry, new antennas and electronics were added to support battlefield integration, and additional sensors and displays were added to support night operations. A major difference between the two systems is the magnitude of consumed *excess* relative to the system's parameters. Table 3.6 is a rough comparison of selected system parameters for the B-52 and F/A-18 showing the B-52 to be significantly larger and heavier. Assuming new components added to each aircraft are roughly the same order of magnitude the relative impact is significantly larger for the Hornet.

An addition to each aircraft of a component weighing 200 pounds (roughly equivalent to upgrading a pair of smart weapons) provides an illustrative example. As a fraction of system weight, this represents a 0.8% increase to F/A-18 weight compared to a 0.01% increase to B-52 weight.

Table 3.6: Comparison of Select System Attributes [54,83]

Aircraft	Height (ft)	Length (ft)	Wingspan (ft)	Weight (lb)
B-52	40.6	159.3	185.0	172,740
F/A-18	15.3	56.0	40.4	24,372

3.6 Chapter Summary

The analysis of lifecycles for the F/A-18 and B-52 in the context of their original design supports the hypothesis that the presence of *excess* contributes to system flexibility.

The analysis of the B-52 demonstrates how the presence of *excess* increased the system's lifecycle value by supporting modifications required to avoid SAMs, carry conventional and precision guided ordinance, and operate in a modern data-centric battlefield. These modifications were all possible without excessive cost due in part to the relative magnitude of consumed excess compared to total excess in the system associated with weight, volume, and range. These

findings support the hypothesis because the presence of ample *excess* is associated positively with flexibility throughout the B-52s lifetime.

In contrast, analysis of F/A-18 demonstrated that a system with little available *excess* (due to challenging initial requirements) was quickly unable to support necessary modifications. The F/A-18 was able to support a limited number of modifications, but rapidly transitioned to experiencing change avalanches for new modifications. These avalanches were driven primarily by the lack of excess in weight, range, and internal volume despite the purposeful incorporation of modularity. The result was an expensive design refresh as a substantially different aircraft. These findings also support the hypothesis by finding that absence of *excess* is associated with inflexibility.

This chapter's primary contribution is providing qualitative evidence of the association between excess and flexibility using historical records and government documents. Evidence from the B-52 analysis finds a positive association between excess and flexibility. Evidence from the F/A-18 analysis finds that a lack of excess is positively associated with inflexibility. The following chapter builds on this study by assessing the relationship between excess and another property of changeability, called robustness. This study is possible to study quantitatively thanks to uniquely well-documented component specifications and the changing requirements associated with them.

Chapter 4: Excess in Gaming PCs

4.1 Introduction

This chapter continues the study of excess by providing quantitative evidence that *excess* can make a system more robust (insensitive to changing requirements) and thereby increase its value. This evidence is drawn from the study from the video game industry data including: console and computer hardware performance specifications with release dates, video game requirements, and contemporaneous expert suggested gaming desktops from 2001 to mid-2019. Additionally, this chapter advances the notion of strategic *excess* (*excess* added to a single component) by examining its potential impact on historical systems. The study of strategic excess provides guidance for how one might baseline the appropriate degree of *excess* inclusion based on technology and requirements trends.

The prior chapter provided preliminary qualitative evidence that *excess* makes systems more changeable, increases system value, and extends service life. However, the exploration of *excess* requires richer models guiding decisions about its form, placement, and quantity. Proof of efficacy requires stronger evidence of *excess'* benefit. Understanding the relationship between excess and value is important because consumers may find the purchase price of products with high excess prohibitive. We also hypothesize that if minimal (or no) excess is included in a system, consumers will see their system lose significant value when requirements change.

Formulating *excess*-based research studies is challenging because data must be collected and analyzed about 1) the design of a system and 2) how requirements change after the system was fielded. Often, the data for most systems is either poorly documented or proprietary. Gaming systems offer a unique testbed for studying and evaluating *excess*. Game developers must communicate the necessary component-capabilities needed, resulting in requirements that are

simply stated with enough clarity that customers can understand if their system meets a minimum threshold. Component capabilities are also readily quantifiable, such as GPU Texture Rate, CPU FLOPS, and RAM memory capacity. Each component's *buffer* accommodates known game releases and partially known future game requirements, whereas *excess* remains for accommodating unknown future game requirements. As discussed in prior research [84], gaming systems become a case study where we can pose the following questions:

- What is the value of *excess* from the perspective of the end user, and is *excess* worth the up-front expense that must be accepted during the initial purchase?
- Should designers create products with built-in *excess*? If so, how much *excess* should there be, and where should it strategically be placed within the system?

4.1.1 A Brief Discussion of Gaming Systems – Computers and Consoles

Two system types are considered in this chapter – personal computers (PCs) used for gaming purposes and consoles. We consider two gaming consoles, the Xbox (designed by Microsoft) or PlayStation (designed by Sony). From the perspective of the consumer, selecting a desktop or console influences what purchasing options exist, the initial performance and cost, system upgradability, and game compatibility between generations. For PC games, hardware performance requirements increase every year. This could be considered a requirements-pull scenario, as playing the latest game may require a system upgrade or replacement. For consoles, manufacturers establish the boundaries of hardware performance for a certain time period (often described in generations of the console – such as PlayStation, PlayStation 2, etc.). This could be considered a requirements-push scenario as game developers must develop their solutions within the confines of the existing machine capability. Consumers typically upgrade their computers every 4-6 years while consoles users upgrade around the 6-year mark [85,86].

Desktop Computers

The desktop computer is designed for modularity and empowers consumer individualization. Purchasing decisions require tradeoffs between performance, cost, and longevity, yielding custom systems that satisfy individualized preferences. Detailed component performance and interface requirements are provided by component manufacturers so that feasibility and performance can be assessed. Computer enthusiasts and industry periodicals help customers navigate the large design space by publishing build guides. These guides contain specific component model suggestions for different build combinations at distinct tiers of performance/cost.

Game producers must also specify relevant component-based performance requirements to ensure that customers can accurately assess their computer's ability to run the game. These requirements have been aggregated into databases that contain requirement data for games dating back 25 years. In this study we simplify a computer by representing it as nine unique component types. We consider four primary components (CPU, GPU, RAM, and storage drive) when assessing if a build satisfies game requirements. The remaining five components support operation and add cost but are not referenced by modern game requirements. These include computer case, power supply, motherboard, sound card, and optical disk drive. We use hardware data and compare expert-suggested desktop build configurations against future game requirements. Given the data available regarding system design and video game requirements, we can assess and study how *excess* in the primary components of desktop computers affects system lifetime value. A description of the functionality of each component is found in Kaif [87].

Consoles

Unlike desktops, console design choices are made by the manufacturer. Storage drive size is one of the only minor customization options available at the point-of-sale. New generations of each console are released every 3-7 years. Purchases made between release windows, regardless of when it is purchased, are the same system with the same components.

Game Options

Often, top-tier games are released for both consoles and desktop computers as game publishers seek to sell as many copies as possible. This suggests that value metrics could be roughly analogous across systems. However, there are several relevant software differences. First, since console hardware is fixed within a generation, video games are tuned/optimized for running on that hardware. Game publishers know exactly what hardware is present and maximize game performance by adjusting game settings or optimizing gaming engines. This may afford consoles slightly improved hardware efficiency compared with an equivalent desktop. Games released for desktops, conversely, have tunable game settings that can be aligned with the available hardware. Users have the freedom of selecting their desired balance between game aesthetics and playability (often measured in frames per second) that best suits their preferences. Newer games can also be played on older machines, though with reduced game aesthetics. Games on desktops are backward and forward compatible so long as minimum hardware specifications are satisfied. For consoles, games from prior generations can be run on a new system, but a game from a new system is incompatible with old generations. Console manufacturers appear to be changing this for the next generation, as discussed in Section 4.6.

4.1.2 Chapter Outline

The primary aim of this chapter is quantitatively assessing whether *excess* improves system lifetime value. We question whether buying a high-end desktop offers more value than a low-end or mid-range desktop when use is expected over multiple years. Data collection and processing are discussed in Section 4.2, and a comparison of desktop and console component-capabilities is presented in Section 4.3. Sections 4.4 to 4.6 focus on the value of excess in a desktop computer, providing quantitative insights into the tradeoff between value and system *excess*. A metric is introduced in Section 4.4 for calculating a desktop's ability to satisfy video game requirements throughout its life. A utility-based assessment of *excess* at the system-level is then conducted for 93 desktop builds. These builds are distributed across three performance/price tiers.

Secondary aims include exploring where and how much *excess* should be incorporated in a system. In Section 4.5, *excess* is considered at the component-level. Specifically, the effect of *excess* is studied when it is added to a single component. An upper bound in system value improvement caused by this allocation of strategic *excess* is developed by calculating how many unsatisfied game requirements are associated with each component. The fractional improvement of playable games from incorporating strategic *excess* is calculated for each component. Evidence is provided showing that strategic *excess* creates meaningful increases in system lifetime value.

In Section 4.6 we analyze the co-evolution of hardware capability and game requirements with the intention of demonstrating how a designer might use historical data for making decisions regarding component *excess*. Analyses of CPUs and GPUs are performed, and insights are drawn regarding what tier of hardware performance satisfies a system lifetime goal without

unnecessary initial expenditure. The paper then concludes with a discussion of the findings and a description of future work.

This research offers insights into how one might baseline the appropriate degree of *excess* inclusion based on technology and requirement trends. Specifically, a quantitative assessment of the relationship between *excess* in desktops and consoles in the context of rapidly improving technologies and the increased demands of video game requirements is provided. This includes explicit assessment of the value provided by different degrees of included *excess*, the placement of that *excess* within the components of a system, and guidance into how designers can use available information when making decisions about the form and degree of *excess*. This knowledge allows engineers and designers to make strategic decisions regarding where *excess* should be incorporated so that value is maintained when operating beyond the system's expected life – a feature that may lead a customer to purchase the system over competing systems.

4.2 Data Collection and Console Comparison

The data collection process includes: 1) collecting game requirements and expert-recommended desktop builds, 2) matching hardware models from suggested desktops and requirements with component-specific performance metrics, and 3) quantifying the number of games each desktop can play in years after it has been purchased (and how well it can play them). A comparison of consoles and desktops capability is then discussed.

4.2.1 Data Collection

4.2.1.1 Game Requirements

Game requirements are how game producers communicate hardware requirements. This information is made widely available at the point-of-purchase (either written on the box or in the product description if purchased online) and is collected in industry publications (e.g. PC Gamer,

Game Informer, Computer Gaming World, etc..) and online databases. The requirements data used in this study are scraped (accessed and saved locally by an algorithm) from the website gamesystemrequirements.com. Data from 5233 games released during the years 1996 through 2020 is captured across 33 information fields that include the publisher, developer, release date, game category (adventure, role-playing, strategy, etc.), average reviews, and hardware performance requirements. Not all fields are present for each game. An abbreviated example is shown in Table 4.1.

Table 4.1: Abbreviated example of data captured for a videogame demonstrating how hardware/software requirements may be communicated

	Game Info				Minimum			
	Developer	**Genre**	**Release Date**	**Reviews**	**CPU**	**GPU**	**RAM**	**Store**
The Witcher 3: Wild Hunt	CD Projekt RED	Action, Role-playing game	2015. May 19 (PC)	Very positive (9.6)	Intel CPU Core i5-2500K 3.3GHz / AMD CPU Phenom II X4 940	Nvidia GPU GeForce GTX 660 / AMD GPU Radeon HD 7870	6GB	40GB

Most manufacturers provide requirements for a “Minimum” setting and a “Recommended” setting. The minimum setting is sufficient for basic game functionality but may result in undesirable effects such as low-resolution game play, lower frames per second, and extended loading wait-times. The recommended settings result in high-resolution rendering, graphical effects such as shadowing and anti-aliasing that notably improve the visual appeal of the game, and reduced load times without a decrease in frames per second.

The standard method of conveying hardware requirements is by referencing a component model/family that satisfies the game’s requirements (e.g. Intel i5-750). Other means of conveying hardware requirements include the direct specification of a performance metric associated with a component (such as the clock frequency of the CPU) or indicating the component’s support for a specific software set (e.g. “GPU must support DirectX 9.0”). Scraped

requirement information is therefore semi-structured data with some natural language and some standard formatting. Translating requirements into performance metrics is discussed in Section 4.4.2.

4.2.1.2 Recommended/Suggested Computer Builds

The second dataset contains recommended desktop builds from 2001 to 2016. Build guides contain component model recommendations and then-current component prices for three tiers of performance. These guides are written for customers building their own desktop who might not be sure what components are "best" for their price tier. The three tiers of performance are Entry, Mid, and Dream. The exact price point for each level varies, but in general the Entry is approximately $500, the Mid at $1000, and the Dream at $2000+. There are two sources for this information: PC Gamer magazine back-issues from 2001-2012 and 2015-2016, and the website newbcomputerbuild.com for data from 2012-2015. Two data sources were necessary because PC Gamer briefly discontinued the monthly article in the studied period.

Data is collected for two build recommendations from each year - mid-year (June or July) and end-of-year, for a total of 93 desktops. Some of the recommended builds include suggestions for peripheral hardware such as mouse, keyboard, monitor, or joystick. This information is inconsistently provided and is generally not a part of a game's requirements. A control volume was established around the computer case. Anything within the boundary of the case was considered, while peripherals were excluded and their costs deducted from the system total. As an example, a 2008 Mid-tier desktop recommended by PC Gamer is shown in Table 4.2.

Table 4.2. A recommended Mid-level build from 2008 with components and corresponding then-current prices from PC Gamer [88]

Computer Component	Suggested Component	Current Price
Case and PSU	Antec P180; 850W PSU	$223
Processor	Intel Core 2 Duo E8500 3.16GHz	$185
Motherboard	ASUS P5N-E SLI	$155
Memory	Corsair 2GB DDR2-800	$50
Optical Drive	Lite-On LH-20A1H	$34
Hard Drive	WD 500GB WD5000AAKS	$60
Soundcard	Creative Labs X-FI Xtreme Gamer	$98
Videocard	Geforce GTX 280	$346
Total Price		$1,151

4.2.1.3 Desktop Hardware Performance Specifications

Specific component models are often reported for game requirements and suggested desktop builds. These models may be from different manufacturers, making comparison challenging without a common performance metric. Performance information for three primary component types (CPU, Graphics Card, and RAM) are collected. In total, information was collected for 1306 CPUs, 1905 graphics cards, and 60 varieties of RAM.

The primary source for CPU and GPU information is a database hosted by techpowerup.com. We discovered instances where a listed component model was not found in the database. In these cases, component metrics were drawn from manufactures' websites or from other data sources as needed.

For CPUs, 15 data fields are captured. The primary fields of interest are model, family, architecture, frequency, number of cores, and release date. The GPU database contains 52 fields of information regarding hardware (number and types of cores/processors, type and amount of onboard memory, interface types), supported software sets (DirectX, OpenGL, OpenCL), and performance specifications (theoretical FLOPS, Texture Rate, Pixel Rate).

RAM specification involves two performance metrics: the transfer rate and the amount of memory present. This information is often aligned with the component name, as shown in Table 4.2. The listed quantity is 2GB, and the transfer rate associated with DDR2-800 is 6400 MB/s.

4.2.1.4 Console Hardware

Specifications for console hardware was collected for the Xbox and PlayStation product families from 2000 to 2017. The release dates, retail price, supported TV resolution, and selected component specifications are shown Table 4.3. Only the GPU texture rate is compared with desktop GPUs since the two are the most comparable. The CPU architecture for some generations of consoles use a different architecture and instruction set preventing a direct comparison with desktop CPUs.

Table 4.3: Specifications for Microsoft and Sony console releases from 2000 to 2019

Release	Model	CPU/APU	Memory Size (MB)	GPU	GPU Texture Rate (GTex/s)	Supported Resolution	MSRP
Mar-00	PS2	Emotion Engine (Sony)	32	Graphics Synthesizer	1.2	480p	$299.00
Nov-01	Xbox	Pentium III (Intel)	64	Nvidia Geforce 3	1.9	480p	$299.00
Nov-05	Xbox 360	Xenon (IBM)	512	ATI Xenos	8	720p	$299.00
Nov-06	PS3	CELL (IBM)	256	Nvidia/Sony RSX	13.2	720p	$499.00
Nov-13	PS4	Jaguar (AMD)	8192	AMD GPGPU	57.6	1080p	$399.00
Nov-13	Xbox One	Jaguar (AMD)	8192	AMD GPGPU	40.9	1080p	$499.00
Nov-16	PS4 Pro	Jaguar (AMD)	8192	AMD GPGPU	131.2	4k	$399.00
Nov-17	Xbox One X	Jaguar Evolved (AMD)	12288	AMD GPGPU	187.5	4k	$499.00

4.2.2 Data Processing

A data processing step is required because the components described in recommended builds and video game requirements must be extracted from their source, matched with

components in the hardware database, and converted into performance specifications. RAM and Storage specifications are given in a standard format. CPU and GPU specifications, however, have a wide variety of formats, shorthand, and specificity that required translation to specific component models/metrics. Examples of these specifications are shown in Table 4.4. Given the volume of data, an algorithm was developed for matching component text with performance specifications. The algorithm works by searching text strings for characters, or words for features. The search order is a balance of the computational effort required and the specificity of the result.

Table 4.4: Examples of videogame requirements before processing

CPU	GPU
Intel Core i5-7400 | AMD Ryzen 5 1400 | or equivalent	512MB 3D Card, Shader Model 3
1 GHz or higher	(Any)
i3 or faster	Shader model 2.0 support
Dual Core	512 MB
Intel 'I' series/AMD K10 series introduced 2009 onwards. 2 physical cores, 2Ghz 64 bit	AMD Radeon HD 7800
Processors with 2.8GHz or great	OpenGL 3.1 Compatible

①"Any" or "None"	②Match component model	③Match specifications
Use the lowest performance component in database	Match names of component models with those in database	• "Dual Core" • ">1.5 GHz" • "OpenCL 3.2 or

Figure 4.1: The sequence of text searches for each CPU and GPU requirements listing

Text parsing for the CPU and GPU uses a four-step process:

I. The search seeks the words "Any" or "None" in isolation. If found, the lowest performance component in the database is used.

II. The search seeks a substring that matches any component in the database. This begins with a search for product family names such as "i5" or "Nvidia Geforce".

a. If a family name is found, a second search using each model in that family is performed. If a match is found, the matched model and performance specifications are stored.
b. Otherwise, the set of possible components is filtered to those in the specified family. Other performance criteria are subsequently used to select the appropriate model within the family such as CPU frequency or GPU onboard RAM.

III. If no component model match is found, the algorithm searches performance specifications. These are component-specific with CPU specifications including cycle frequency and number of cores. GPU specifications include OpenGL version, OpenCL version, DirectX version, and onboard RAM quantity. If one or multiple of these are found, they are used to filter list of possible model matches. After the performance specification search is complete, the component in the filtered set and the lowest performance is selected.

IV. If no matching criteria are found, the associated game and performance level (Minimum or Recommended) pair is not included in the cleaned database. Of the original set of 5233 games, 293 games are removed from the study (4940 remain).

4.3 Console and Desktop Comparison

While the focus of this chapter is on the value (and placement) of *excess* in a desktop, comparing consoles and desktops is prudent because it highlights the difference between *buffer* and *excess*. Console manufacturers must estimate the extent by which game developers can make use of the technology placed within a certain console generation. They must also estimate how this capability will be consumed over time by game developers. Console game design is driven by a push cycle, as the console manufacturer establishes system requirements – establishing the *buffer* within each component - which are then passed onto the game developers.

Desktop game producers are not directly constrained by the requirements established for them by a third-party. Game producers balance the capabilities offered by new technologies against the expected capabilities of the desktops owned by individuals (the consumers of the game). This creates a pull cycle for game design, as advances in component technology drive the advancement of game requirements. As new desktop components are constantly being released, desktop owners purchasing a new system get the best technology available within the budget of their performance tier. Desktop owners also have the freedom of updating components within their system. Yet, how well their system can play games in the future is less uncertain, as game developers have significantly more freedom. This requires the allocation of *excess*.

GPU performance for consoles (shown in Table 4.3) is compared against the recommended desktop builds in Figure 4.2. The y-axis is the texture rate for each GPU and is presented on a log scale. Console GPU performance is held constant between releases since the GPU remains unchanged within a generation. The first two generations of each console had GPUs that performed comparably with the Mid desktop. In 2014, the next generation of console GPUs had performance slightly below the Entry desktop. These consoles were quickly refreshed, and the new GPU performance was comparable with the Mid desktop.

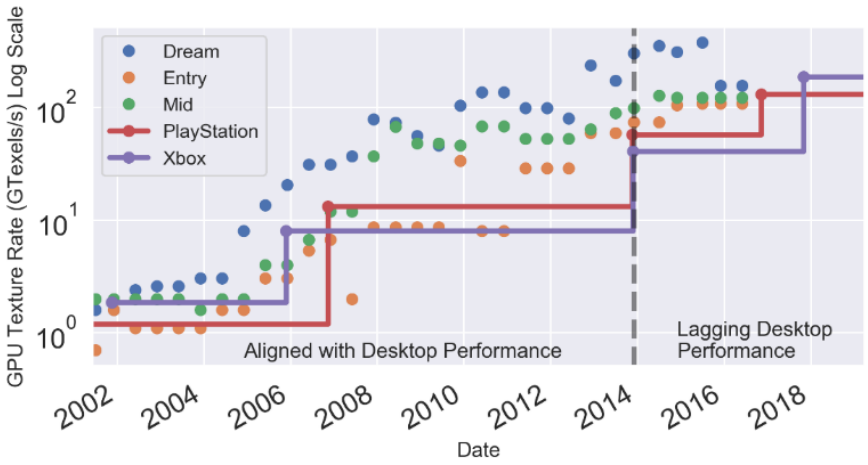


Figure 4.2: Comparison of console and computer GPUs performance showing computers' more frequent releases versus longer Xbox and PlayStation generations

While console performance aligns with entry/mid builds at launch, desktop game requirements are constantly evolving. Using the third quartile of desktop game requirements as a surrogate for "most" desktop games, console GPU performance is compared with minimum and recommended desktop game requirements in Figure 4.3. A new console satisfies the recommended requirements for all third quartile desktop games at launch. As time passes, desktop GPU game requirements increase, and the consoles no longer meet recommended requirements. Console GPU performance has also fallen well below the GPU performance of the latest Entry desktop. Consoles are refreshed when their performance aligns with the third quartile minimum desktop game requirements.

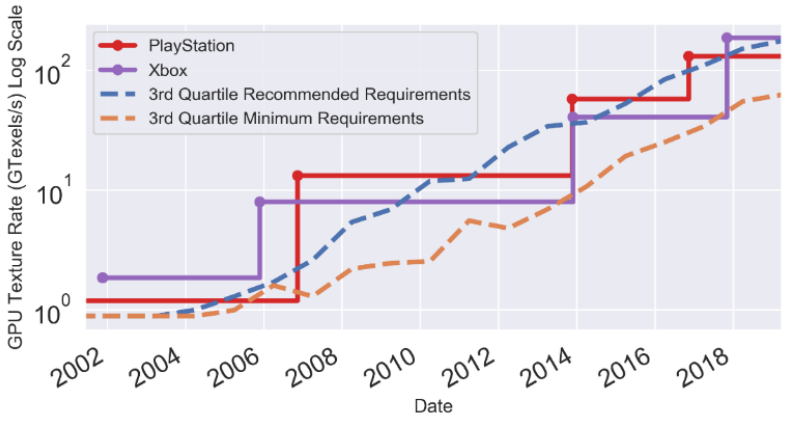


Figure 4.3: Comparison of console and minimum game requirements for GPUs showing console showing consoles capable of supporting most game GPU minimum requirements for the entire generation and most recommended game setting for about half the generation

A consequence of a push cycle is that console manufacturers must decide when a new system should be released, presenting a tradeoff of going with an available GPU or incorporating a better GPU in the near future (allowing console GPU performance to fall even further behind desktops). Microsoft and Sony took different business strategies. Sony waited an additional year when releasing the PlayStation 3. While the Xbox 360 was available a year earlier, it had worse GPU performance than the PlayStation 3 from 2007 until 2013. Assuming a GPU comparable with a Mid-tier desktop build, one additional year of waiting for Sony would have resulted in four-fold better performance from 12 to 48.2 Gtexels/s. Generational refresh timing is critical for maintaining console competitiveness. Optimization by game developers when porting desktop games to consoles may marginally improve game performance, but anecdotal evidence suggests that customers experience decreasing enjoyment of new games played on the console [89–91].

The infrequent release of consoles requires that decisions about *buffer* follow a push cycle driven by desired end-of-generation game requirements. At the beginning of a new generation there is significant *buffer* as developers are only beginning to leverage the capabilities of the new components. This *buffer* is then consumed, with no pathways for in-generation

modification. However, desktops are frequently released, and future game requirements are much less certain. The performance tiers have significant implications on how much *excess* exists in a newly purchased system. The constant release of new technologies creates a pull cycle that drive increased gaming requirements. Decisions about *excess* must now be made by the consumer: 1) what desktop performance tier offers the best value, and 2) is there value in upgrading a component at the time of purchase? These are explored in Sections 4.4-4.6.

4.4 Desktop Excess Assessment

The data processing step discussed in Section 4.2.2 linked game requirements and recommended desktop builds to specific component models. These component specifications determine whether a desktop meets the recommended or minimum game requirements. There are multiple games released each year, and multiple years of games are considered in this study. We introduce a System Performance Metric (SPM) that scores how well a desktop meets minimum and recommended game requirements for games that are published within six years of a desktop being purchased.

4.4.1 Component Specification Comparisons

Designing a CPU or GPU requires tradeoffs that allow the component to perform better for some tasks at the expense of others [92]. The standard practice is benchmarking component performance at specific tasks. Benchmarks include metrics such as framerate when rendering a specific game or computational time for compressing a file. These results can be found in online databases such as cpubenchmark.net and userbenchmark.com. Unfortunately, these databases are not sufficiently comprehensive, as data is lacking for many component models, especially those used in the earlier years that define our study. While not ideal, raw performance specifications are used. The specifications used for comparing each component type is tabulated in Table 4.5.

Component	Specification	Units
CPU	Theoretical Floating-Point Operations per Second	GFLOPS
GPU	Onboard RAM	GB
GPU	Theoretical Texture Fill Rate	GT/s
RAM	Quantity of Memory	GB
Storage	Quantity of Memory	GB

Table 4.5: Performance measures used to compare a desktop computer to game requirements.

4.4.2 SPM Calculation

The System Performance Metric captures how well a desktop build meets future video game requirements. That is, we consider only games that are released in the same, and future, year(s) after the system has been purchased. This measure ranges from [0–1] on a unitless scale. A score of 0 indicates that no future games are runnable. When calculating the SPM, games are first binned based on release date in 12-month blocks. Games released in the 6 years after the desktop build was published are considered, as consumers typically upgrade their desktops every 4-6 years [85,86]. By choosing this threshold, the range of desktop builds assessed is limited to those prior to mid-2013.

Each game is assigned a score, u_i, based on which requirement set is satisfied by the desktop. The following rubric is used for assigning a score to each game:

- A score of 1.0 is assigned if the desktop satisfies recommended game requirements.
- A score bounded between 0 and 1 is assigned if the desktop satisfies:
    - minimum game requirements but not recommended requirements,
    - OR the game only has minimum requirements published which are satisfied.

  This score reflects that the customer gets some enjoyment from playing the game. In this research, a value of 0.5 is used. Exploring possible ways of assigning u_i per game when recommended requirements are not met is a source of future work.

- A score of 0 is assigned to any game where the desktop does not satisfy minimum requirements.

Individual game scores are aggregated within each 12-month block, as shown in Equation 4.1, resulting in a bin total. In this equation, r_i is the relative importance of each game, n is the number of games released in a year, and j is the year considered. For this study, the relative importance of each game was set to 1. Exploring how r_i changes per user or market segment will be explored in future work.

$$(Bin\ Total)_j = \sum_{i=1}^{n} r_i u_i \tag{4.1}$$

The average of this bin is also calculated, as shown by Equation 4.2. This procedure for binning and averaging was chosen because some years have more game releases. This procedure prevents skew caused by bin count size. As shown by Equation 4.3, the SPM is calculated by averaging the bin average for six consecutive years. This metric allows for desktop build comparison across performance tiers and across time. A notional example is presented in Table 4.6, where only three years and 11 games are presented for brevity.

$$(Bin\ Average)_j = \frac{\sum_{i=1}^{n} r_i u_i}{n} \tag{4.2}$$

$$SPM = \sum_{j=1}^{6} \frac{\sum_{i=1}^{n} r_i u_i}{n} \tag{4.3}$$

We begin by exploring the primary hypotheses that *excess* can improve system value by calculating the SPM for each suggested desktop build. The three tiers of suggested desktops (Entry, Mid, and Dream) are used as proxies for different levels of *excess* inclusion.

- The Entry system is the baseline and is assumed to have little or no *excess*.
- The Mid system is assumed to have moderate levels of *excess*.
- The Dream system is assumed to have significant levels of *excess*.

The expectation is that desktops with more *excess* will have superior SPM scores. In Section 4.4.4, the analysis focuses on whether improved SPM scores enabled by *excess* can be justified when compared against system cost.

	Year 1		Year 2		Year 3	
	Game 1	1	Game 5	0.5	Game 7	0
	Game 2	0.5	Game 6	1	Game 8	0.5
	Game 3	1			Game 9	0
	Game 4	1			Game10	1
					Game11	0.5
Bin Total	3.5		1.5		2	
Bin Avg.	0.88		0.75		0.4	

SPM Value:	0.675

Table 4.6: Notional SPM calculation for three-year period**.**

4.4.3 Desktop Performance Using the SPM Metric

As a direct comparison to the data presented in Figure 4.3, the GPU performance of suggested desktop builds is presented against game requirements in Figure 4.4. As before, the third quartile of recommended and minimum GPU game requirements is used. The Entry build generally has a GPU Texture Rate that is slightly above the recommended game requirements when it is released. However, GPU performance drops below the recommended requirements within the first 2-3 years of release as game requirements increase. The Mid and Dream builds exceed the recommended game requirements and continue to do so over a longer multi-year period. The SPM for each performance tier (for a computer built during the given time period), is

plotted and shown in Figure 4.5. Recall that two builds are considered during each calendar year. The values are co-plotted with a moving average for trend identification.

Figure 4.4: Suggested computer GPU performance vs Video Game Requirements showing desktop hardware increasingly outpacing game requirements beginning around 2005

Figure 4.5: Suggested computer build SPM values over time demonstrating more excess provides robustness vs requirements changes and a trend of increased game requirements lag in the latter portion of the study period

From Figure 4.5, the SPM is higher for Dream and Mid desktops. This supports the hypothesis that greater levels of initial system *excess* correlates with improved game requirement satisfaction over the life of the system. The mean SPM for the Dream system is 0.57 out of a maximum value of 1. This is compared against a mean SPM of 0.42 for Mid builds and 0.25 for

Entry builds. A larger mean SPM means the increased ability to run more games at the recommended game settings and a better gameplay experience.

A temporal trend is observable. The SPM for all desktops follows an upward trajectory. Further, the SPM for all desktops prior to 2005 is lower than those after 2009 by an average of 31%. This suggests that computer game requirements lag hardware development, especially during the latter period of the study, and that the hardware capabilities pull game requirements upward. The disparity between SPM values for each desktop type also shrinks over time. In 2005, the Mid and Entry SPM values are 37% and 23% of the Dream value respectively. By 2013 the gap narrows to 91% and 69%. This gap size has implications for the value of the added *excess,* as the Mid and Entry computers cost 25-50% and 15-25% of the Dream, respectively.

In the early 2000s game developers were creating games that quickly outpaced the capabilities of older computers. By 2007-2008, component improvements begin outpacing game requirement increases and SPM scores improve. This results in a SPM gap reduction between the Mid and Dream builds beginning around 2006 and the Mid and Entry builds beginning in 2011. If the components in a lower tier system can achieve a SPM value closer to 1, as seen in 2012 and 2013, incorporating additional *excess* offers less room for improvement as SPM has a maximum value of 1. This raises questions about the value of *excess,* an analysis that requires the consideration of benefit and cost.

4.4.4 Assessing the Utility of a Desktop Gaming Computer

SPM only accounts for game playability. If the goal is maximizing SPM, the Dream build is always the best option, as shown in Figure 4.5. Consider that case as Scenario 1, where a customer is only interested in maximizing SPM. However, *excess* also comes with increased cost. By considering an additional metric (cost), addressing the relative importance of SPM and

cost becomes necessary. A definitive analysis of the optimal combination of SPM and cost requires calculating a utility value [40,93] after modeling customer preferences. In this research, however, the goal is demonstrating that *excess* can increase system value – we are not determining what that value is. We pose two additional scenarios illustrating the value of *excess* under different hypothetical customer preference structures.

Scenario 2: Balance SPM and system cost

The utility function of this scenario is the ratio of SPM and initial system cost for each build, i, as shown in Equation 4.4.

$$Utility_i = \frac{SPM_i}{(Initial\ system\ cost)_i} \tag{4.4}$$

The resulting values are plotted in Figure 4.6. The utility of the three computers is inverted - the Entry build generally performs the other two builds. Unlike in Figure 4.5, the increased *excess* of the Dream and Mid builds is not worth the extra cost.

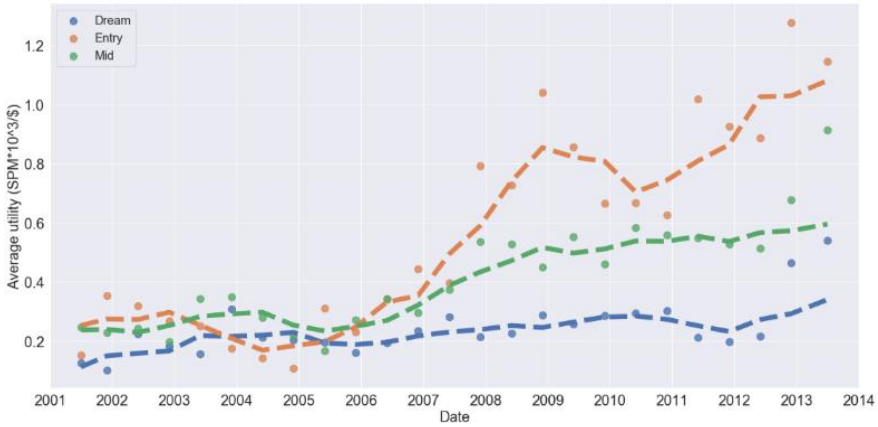


Figure 4.6: Utility using a ratio of SPM to cost demonstrating the entry level system generally outperforming Mid and Dream computers.

We also observe that in the early 2000s, the Mid and Dream builds outperform Entry builds (between mid-2003 and mid-2006). Yet, as time progresses, the:

- Dream build values are relatively constant.
- Mid build values approximately double.

- Entry build values increase by approximately a factor of 4.

Keeping in mind target build costs, the observed trends are driven by the performance gap decrease plotted in Figure 4.4. In the early 2000s, the performance increase associated with the higher-end builds warrants the 2 and 4-fold increase in cost. As the Mid and Entry builds close the performance gap, the extra performance of the Dream build does not compensate for higher costs.

Scenario 3: Initial cost and SPM associated with high-end games

The utility function of this scenario is constructed using initial costs and the SPM associated with the cutting-edge, high-performance games. We consider the 20 games from each year with the highest requirements. Using the complete bin of games released each year, we normalize GPU, CPU, and RAM requirements on a 0-1 scale. The normalized GPU, CPU and RAM values are then averaged. As shown in Equation 4.5, the utility function is the ratio of this SPM-20 value and the initial system cost for each build, i. The resulting SPM is divided by the cost of the system. The resulting values are plotted in Figure 4.7.

$$Utility_i = \frac{(SPM - 20)_i}{(Initial\ system\ cost)_i} \tag{4.5}$$

By considering the 20 most resource intensive games, the Dream and Mid builds increase in utility relative to the Entry builds. The increase is sufficient for the Mid builds to have a higher utility than the Entry build in most years.

Figure 4.7: Build utility using only top 20 performance games in each year demonstrating Mid computer generally having the highest utility.

The outcomes from these scenarios highlight the importance of considering *excess'* benefits and cost. When only considering SPM, the Dream build is optimal. When cost is included and a tradeoff analysis is required, the Entry or the Mid build may be optimal. The overall conclusion from this analysis is that *excess* does enable better system lifetime requirement satisfaction (at additional extra initial cost), which can provide more utility depending on customer preference. This provides quantitative evidence that *excess* can improve system lifetime value, while also showing that *excess* must be strategically allocated within a system.

4.5 Strategic *Excess* at Component Level

From Figure 4.5, most desktops cannot play every game released over the 6-year window. In this section, we assess which component(s) are at fault for a desktop's inability to satisfy game requirements. Our goal is understanding how strategic *excess* (*excess* added to a single component) can improve requirement satisfaction. The principle of strategic *excess* is that increasing a minor subset of component specifications can yield significant system improvement at marginal cost. Here, adding *excess* means buying a higher performing component than recommended in the build guide.

Employing strategic *excess* requires (1) knowledge of which component(s) could contribute to a desktop's inability to meet future requirements, and (2) the fractional improvement realized by upgrading said component. This section quantifies the extent to which each component contributes to unsatisfied future requirements. The improved SPM score when replacing the most limiting component in the desktop is calculated.

Unsatisfied requirements are divided into 5 categories: one for each component and one multi-component category. If a desktop does not meet requirements because of multiple components, that game is only placed in the "2+ Components" category. This category is important because strategic *excess* in a single component will not allow these games to run.

The evaluation steps are: 1) binning all the suggested computer builds by deployment year, 2) tallying the components responsible for unsatisfied game requirements in future years for each desktop, and 3) dividing the result by the total number of possible runnable games. This evaluation is performed for each of the primary components. The value for each component category is calculated using Equation 4.6. The variables are defined as follows: i is the one of the n desktops in a given year, j is a game in the set of all games (k) in desktop *i's* 6-year lifetime. $R_{i,j}$ is true if game $j's$ requirements are not met for desktop i because of that component.

$$\frac{\sum_i \sum_j \begin{cases} 1 \; if \; R_{i,j} = True \\ 0 \; if \; R_{i,j} = False \end{cases}}{\sum_i \sum_j 1} \tag{4.6}$$

The resulting proportions of component(s) responsible for unsatisfied game requirements for all desktops are reported by release date in Figure 4.8. The downward trend in the total fraction of un-runnable games over time is expected, given the upward SPM trend shown in Figure 4.5. The most common category responsible for unfulfilled requirements is "2+ Components." The percentage of un-runnable games in this category is 59% for Entry, 42% for

Mid, and 25% for Dream. However, strategic *excess* does provide significant opportunities when only one component fails in meeting a game requirement.

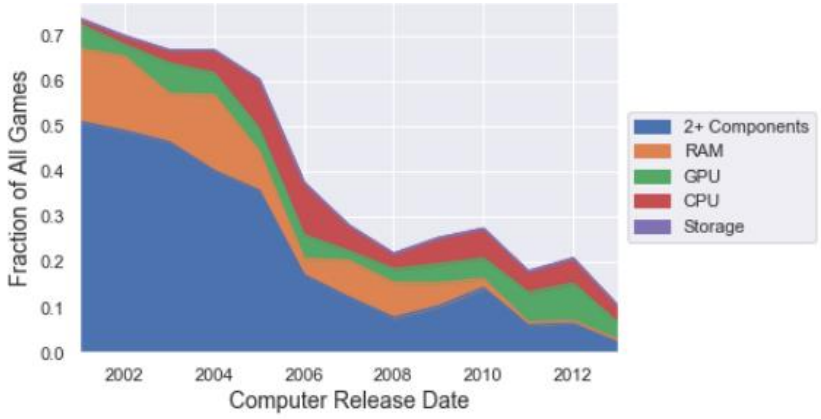


Figure 4.8: Fraction of unsatisfied game requirements per component by computer type and year demonstrating that 2+ component failures are the most common and that the total un-runnable fraction decreases over time for all systems

When looking at single requirement failures, lack of RAM is the largest initial culprit. At time progresses, the CPU drives single requirement failures, and by the end of the study, lack of GPU *excess* causes the greatest failure in game requirement satisfaction. We evaluate the upper bound for system improvement via strategic *excess*. This is performed by improving one desktop component, per build, at a time. We assume that this improvement negates all instances of unsatisfied game requirements from that component for that desktop. Results are plotted in Figure 4.9 by considering desktop type and the component modified.

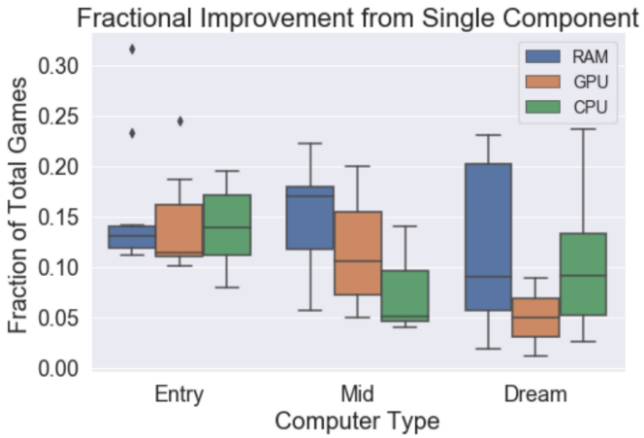


Figure 4.9: Fraction of games made runnable by adding strategic excess by component and computer

The result of this upper bound analysis is that strategic *excess* yields a moderate improvement in SPM. This change is more pronounced in the modified Entry system, as it can now run 15% more games. This is compared against a 13% improvement for the Mid and 11% for the Dream. This is important because strategic *excess* offers an opportunity for moderately improving system performance with only a marginal increase in system cost.

For example, the average cost of RAM is 7% of the total system cost. Yet, increasing the amount of RAM can increase the number of runnable games by 14%. Upgrading this relatively low-cost component can result in a considerable system performance return. This is significant as it serves as an economically efficient alternative to purchasing a higher-tiered system.

4.6 Temporal Comparison of Hardware Performance vs Requirements

Excess may not be beneficial if requirements change gradually relative to the life of the system once inclusions costs are considered. This is also true if requirements change much too quickly and rapidly consume all available *excess* [17]. We examine the technology trends underlying the computer gaming market so that we can assess how much *excess* was appropriate. Specifically, we GPU and CPU and contrast component capability against video game requirements.

A comparison of GPU capability vs. GPU game requirements is presented as a semi-log plot in Figure 4.10. The mean and 95th percentile of GPU performance and game requirements are calculated using quarterly-binned hardware/game releases. From this data we find that hardware performance and software requirements increase at approximately the same rate, though there is a lag between hardware release and software requirements. For both the 95th percentile and mean lines, game requirements (solid lines) are right-shifted by a few years relative to hardware performance (dashed lines). We hypothesize that videogame developers require a few years to fully utilize new hardware when creating game features. This suggests a paradigm of technology driving innovation rather than market demanding technologies to fulfill needs [94].

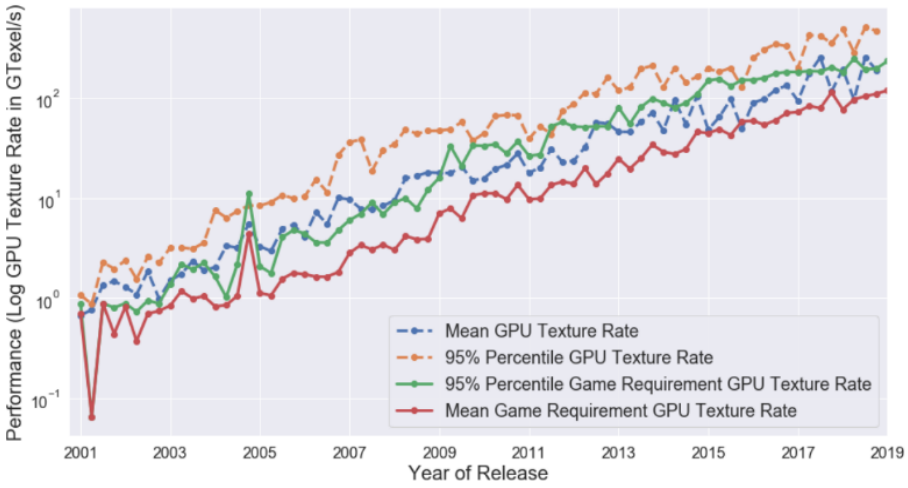


Figure 4.10: Mean and 95% percentile of GPU hardware releases and game requirements binned quarterly demonstrating an exponential increase for both

We present an analogous plot for CPUs in Figure 4.11. CPU releases are less frequent than GPUs and are binned annually. We observe a larger lag by game requirements suggesting that game developers are slower at fully utilizing CPU capacity. A notable decline in CPU game requirements occurs after 2011. While CPU capability and game requirements increase at approximately the same pace (pre-2011), game requirements begin leveling while hardware

capability improves. The implication is that the return on investment of CPU *excess* is reduced after 2011.

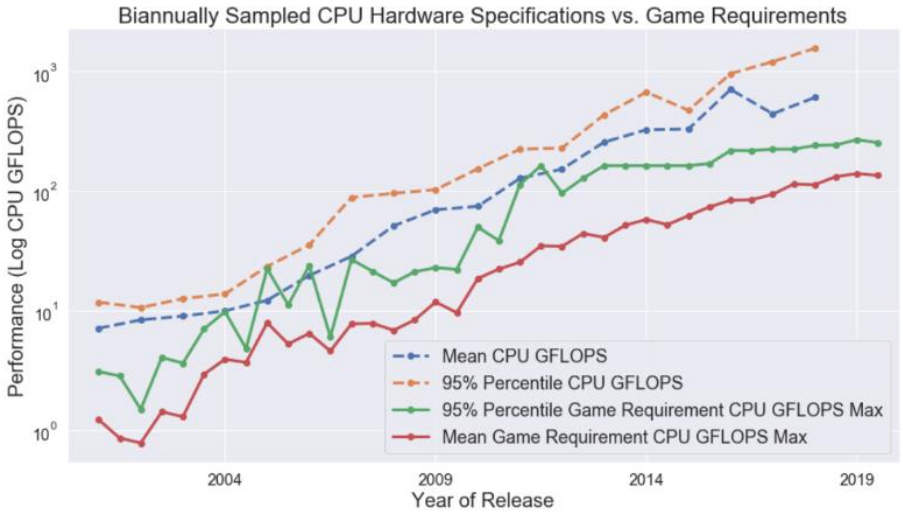


Figure 4.11: Mean and 95% percentile of CPU hardware releases and game requirements binned annually demonstrating an exponential rate of increase for both requirements and hardware with game requirements leveling off around 2013

We also estimate a component's useful lifetime by calculating the number of years between hardware release and when videogame requirements reach the same level of performance (CPU GFLOPS or GPU Texture Rate). Maintaining the desired system-life of 6 years established in Section 4.1.1, useful component-lives shorter than 6-years limits system value. These components did not have enough *buffer* to accommodate a projection of game requirements over time. Conversely, component-lives longer than 6 years have *excess* and incur additional initial system costs. Discrete tiers of component performance are established by selecting values representative of different quantiles from all components released within a given year. The duration of acceptable performance (or component life) is calculated by counting how many years the performance metric associated with the component exceeds the median game requirement (binned annually). The series have differing length because the final plottable datapoint occurs when present day game requirements do not exceed the hardware performance

value. We illustrate the approximate component life for different performance tiers in Figures 4.12 and 4.13.

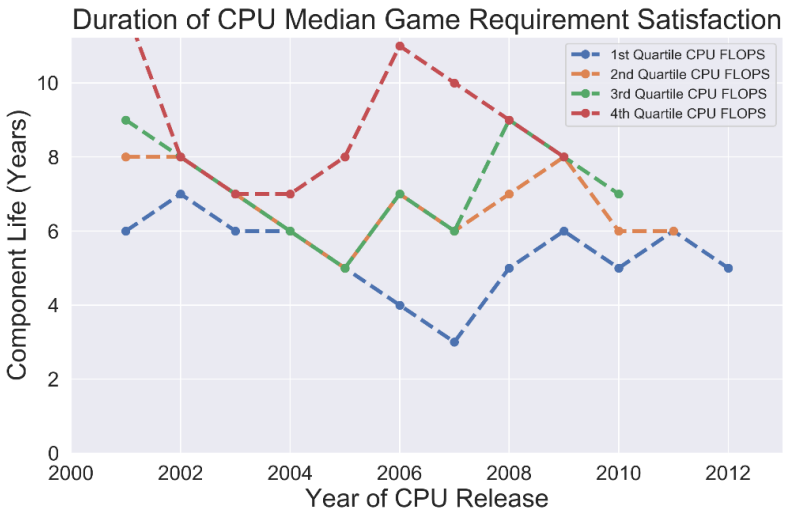


Figure 4.12: Comparison of levels of CPU performance and ability to satisfy median game requirement demonstrating between 1st and 2nd quartile generally meeting 6-year lifecycle target

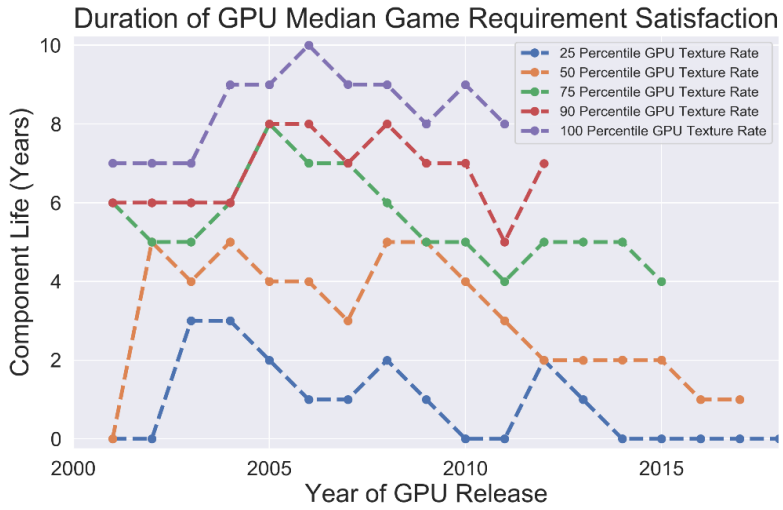


Figure 4.13: Comparison of levels of GPU performance and ability to satisfy median game requirement demonstrating between 75 and 90 percentiles generally meeting 6-year lifecycle target

Lower-quantile CPUs are generally sufficient for a 6-year lifespan. The general trend for CPU lifespan is convex – the effective life of early CPUs decreases until 2005-07 and then increases. GPUs, conversely, have much shorter lifetimes. Only twice does the 1st quartile GPU reach 3-years of useful life. There are 9 instances where the GPU is insufficient in the year it was

released. Only the 75th and 90th percentile GPUs have a 6-year effective life. This strongly indicates that the GPU is a component where *buffer* and *excess* is warranted (and likely desired). It would also be reasonable to select a lower quantile CPU so that more money could be spent on a powerful GPU, demonstrating that *excess* allocation should be done strategically.

4.7 Discussion

The research presented above represents the first quantitative study of excess and its effects on system performance and value using historical data for both changing requirements and technology advances. This analysis lays the groundwork for the integration of excess modeling into holistic design processes like Value Driven Design [95] and Decision Based Design [40] by demonstrating that excess can improve both desktop performance and the utility derived from the improved system, showing how strategic excess can be used to improve system performance with minimal cost, and then demonstrating how historical technology performance and requirement trends can be used to identify the tier of component performance likely to satisfy requirements for a desired system lifetime.

Section 4.3 compared console hardware against both comparable recommended desktop computers and desktop game requirements. There are three key observations from the analysis in Section 4.3. At launch, console GPU performance ranges from slightly less than Entry to better than the Mid computer. Consoles are also priced comparably or less than an entry level computer. Second, from a technology perspective the fixed nature of console hardware is riskier. A large change in component performance soon after generation launch can make a console far less capable than systems with shorter generations. These systems are also not upgradable. Finally, at generation end, console games are not state-of-the-art, suggesting that owner utility

will diminish. These observations have implications for customers, game developers, and retailers.

From the customer's perspective, the excess available in a console in the year of its release is approximately the same as an Entry level computer. As time passes, console excess is consumed while a computer can be upgraded or new a model can be purchased. A console generation within its first year or two of life may offer long-term value, yet this value is significantly reduced if a purchase is made near end-of-life.

Developers must specify performance settings and optimize configurations for each generation. An additional burden occurs when releasing games near new generations. The developer must either expend the resources to optimize the game for both systems, release a lower quality version playable on the old system (reducing future sales), or design for the new generation only and forgo sales on the legacy system.

For retailers, the infrequent and large changes in console performance between generations leads to a boom and bust cycle. Sales pick up after a new console is released but are reduced when new generation is imminent. As an example, investors are questioning whether the retailer GameStop can survive the reduction in sales leading to the 4th quarter 2020 console releases [96].

Ultimately it appears that console manufacturers have recognized that the current model is not optimal. The consoles released in 2017-2018 (Xbox One X and PlayStation 4 Pro) are both incremental improvements released only 4 and 3 years since the prior generation. These systems feature improved hardware, but the same system architecture allows for both forward and backward game compatibility [97]. Microsoft has signaled its intent to change its console release schedule entirely following the release of the Xbox X in 2020. [98,99]. Future console releases

will offer incremental improvement and more frequent release dates while maintaining forward and backward game compatibility.

Game requirements may not be the only factor driving console and desktop performance. Each system must be connected to an external display, typically a television for a console and a monitor for a desktop. Focusing on the console, television resolution increased during the study period requiring more computation from the GPU for higher resolutions. There have been four generations of TV resolution during the study period as shown in Table 4.7. In 2002, consoles supported SD televisions which were soon surpassed in sales by SHD around 2004. The second generation of consoles in 2005-2006 supported SHD which was surpassed in sales by FHD around 2009. This was halfway through each console's generation of the time. Console manufacturers appear to be targeting the most recent generation of televisions which (in theory) are what customers are most likely to have. The need to present games on higher definition displays is therefore be another external driver for hardware performance. Connected systems (or other external factors) may also contribute to console and desktop requirements. While the primary driver of system requirements is likely game requirements, a full excess assessment should consider these other externalities.

Table 4.7: Commercial TV attributes, dates of introduction, and majority of sales

Name	Resolution (Vertical x Horizontal Pixels)	Total Pixels (Millions)	Introduction (Approx)	Majority of Sales (Approx)
SD (480p)	720x480	0.3456	Prior to study	Prior to study
SHD (720p)	1280x720	0.9216	Prior to study	2004
FHD (1080p)	1920x1080	2.0736	2003	2009
UHD (4k)	3840x2160	8.2944	2013	2017

Sections 4.4 to 4.5 focus solely on desktops since the customer exert far more control of *excess* placement. Three different preference scenarios (trading between performance and cost) were presented. In the first scenario the customer was indifferent to cost and only desired

performance. In this case the Dream tier dominates the lower tiers. In the second scenario the customer cares about both price and performance, with utility represented by the ratio of performance to cost. We found that the optimal tier is dependent on the specific year chosen. Entry builds dominated all years, except between 2003-2006. During this period, the Mid build was optimal. In the final scenario, the customer cares about cost and the ability to run the 20 games per year with the most stringent requirements. In this scenario Mid builds are the optimal choice 44% of the time, Entry builds 36%, and Dream builds 20%. The results from each scenario indicates that *excess* does provide robustness to requirements changes, but that the value of that *excess* depends on customer preferences.

In Section 4.5 we explore the notion of strategic *excess* as the customer's ability to improve excess placement within the recommended desktops. We found that the largest category of unsatisfied requirements (42%) was caused by 2+ components. An analysis of all the single component categories found an average of 13% improvement in requirement satisfaction from strategic *excess*. Strategic *excess* in RAM specifically resulted in a 14% improvement in requirements satisfaction from a component that, on average, accounts for 7% of system cost. This analysis demonstrates that strategic *excess* does increase robustness to future requirements with marginal increase in cost. By performing this type of analysis, engineers and designers may reconsider how new system requirements are elicited and how system value may improve by over satisfying current requirements. This research offers an analysis of requirements that was not previously considered and can transform requirements practices in industry.

Finally, Section 4.6 describes how a customer or designer may use historical requirements and technology trends for selecting what tier of performance should be selected for a new desktop's components. The lifetime for discrete tiers of CPU and GPU performance was

assessed by comparing the components' performance metrics with the median game requirements in future years. We found that a 2nd quartile CPU performance was adequate for a 6-year life, while a 3rd quartile or 90th percentile GPU performance was needed. These estimates demonstrate how trends of hardware performance and software requirements over time can be used for informing baseline performance for each component when building a new computer.

It is important to note the limitations of this study. The authors selected computers and consoles as the topic for this study as they are conducive to *excess* analysis. This is due to the well-defined contribution of each component, published prices, and the discretization of said components (note the discretization of components do not change over time). Many electromechanical systems do not share those benefits as the system architecture changes significantly over time. For instance, consider the transition from internal combustion to hybrid to electric vehicles, where the analysis of *excess* may be challenging. Nonetheless, this does not discount the findings of this research; that *excess* has a role and presence in the design of systems.

4.8 Chapter Summary

Prior research has championed the advantages that changeability brings to a system [15]. The ultimate goal for this research is supporting the transition of changeability from something "vague and difficult to improve, yet critical to competitiveness" [100] into a measurable property consistent with utility and value based design methodologies using the concept of excess. This research provides a step in that direction by assessing: by whom decisions regarding excess placement are made, what the implications of those decisions are, and by quantitatively evaluating the impact of *excess* on a system's ability to satisfy future requirements all using the unique opportunity afforded by the availability of historical information about computer

hardware and video game requirements. Specifically this chapter: 1) compared console and desktop hardware decisions assessing the benefits and drawback of each 2) evaluated the impact of different computer *excess* tiers for satisfying future video game requirements, 3) assessed the prospect of using strategic *excess* to increase the computer's ability to satisfy future requirements, and 4) compared trends in hardware performance and game requirements to suggest a baseline tier of component performance for new computer construction.

This research in this chapter suggests that mass production and the associated economies of scale represented by the console perform worse than more custom desktops unless the console is purchased near the release of a new console generation. This is because consoles do not refresh their components (and thereby the system excess) as frequently as desktops. The anticipated transition to a more PC-like release model by console manufacturers reinforces this notion. Excess also supports customization in a valuable way. Expanding the number of systems on offer from one (the console) to three tiers of recommended desktops (with different degrees of excess) allows a consumer to better match their preferences with a system. A consumer with a strong preference for high-performance and an insensitivity to cost will prefer a higher end system with more excess and vice-versa as was shown. Additionally, consumers may further customize their desktops beyond the expert recommendations with strategic excess further improving system performance under ideal circumstances.

In total, this chapter provides evidence for the value of *excess* and provides a starting point for future changeability and *excess* research ideally providing designers with new means of incorporating changeability in addition to modularity. While this chapter demonstrates that excess can improve system lifetime performance and can improve system lifetime utility a generic model of excess and changeability is needed. This model should include how *excess*

should be represented at the component level, how uncertain requirements should be evaluated, and how the presence of *excess* interacts with change propagation and improves system changeability. Importantly, there are also connections between excess and modularity that should be explored. For instance, if a consumer buys a desktop and then upgrades individual components on an ad hoc basis, how does that change the optimal initial component selection? A desktop is very modular, requiring little effort to replace individual components, so what limits desktop life? What is the optimal timing for replacing components and which component performance tier is best? These are the fundamental questions the Chapters 5 and 6 seek to answer.

Chapter 5: Dynamic Change Probabilities and their Role in Change Propagation

Evidence from Chapter 3 supports the notion that long-lived systems are likely to experience many independent modifications during their lifecycles. Prior literature provides tools for predicting how a change in a fixed system is likely to propagate as discussed in Chapter 2, but these tools do not address change propagation across multiple, independent modifications. The phenomenon of a modification consuming *excess*, thereby increasing the likelihood of change propagation in future modifications, is studied in this chapter as Dynamic Change Probabilities (DCP). This research builds on change propagation techniques, network theory, and *excess* to provide high-level guidance about how DCP may alter change propagation within a system over time and what the ramifications of those changes might be. A sample of existing and synthetic systems are explored, as we show that the rate of change likelihood increase following a modification depends on the number of components (nodes), the dependencies between components (edges), and initial change propagation probability values (edge weights). Results also show that *excess* placement in specific components, and the presence of system hubs (high degree components), can mitigate the impact of excess consumption when multiple system modifications are made over time.

5.1 Introduction

When new or changing requirements emerge, the anticipated design and manufacturing effort influences engineering decisions about whether an in-service system should be modified or left unchanged [15]. The modification of a single component often propagates from that component to other components (connected via design dependencies), resulting in additional secondary supporting modifications that further complicate desired system changes [19].

In response, researchers have developed approaches that give engineers insight into how design modifications might propagate throughout a system. Such methods include the Change Propagation Method (CPM) [23], Design for Variety (DfV) [24], and network theory approaches [37]. A generalized description of existing change propagation methods is:

- discretize the system into components at the desired resolution (from component level to subsystem level)
- identify dependencies between the components
- assess the strength of each dependency (i.e. the likelihood that change may propagate along the dependency)
- assess the assembled model providing metrics for the likelihood or risk of change.

In this research, we focus on the third and fourth steps as we assume a representation of the system can be created and dependencies identified. Step 3 in particular is difficult to assess in a rigorous way. Existing techniques either skip step 3 by using unweighted edges or elicit expert opinion (as in CPM and DfV). Embedded in the expert responses are assessments of what new or altered requirements may be imposed, how the modifications might be implemented, and how likely it is that components linked by dependencies require supporting modifications.

Available design margin (called *excess* based on the work in [49]), based on the present state of the system, is implicitly included in the expert assessment of design dependencies. Existing research has shown that *excess* "allow(s) the component to grow or to be moved and can therefore absorb potential changes to the product" [16]. If a modification consumes *excess*, the assessment in step 3 must be updated so that it reflects the new state of the system. That is, reduced *excess* increases the likelihood of change propagation.

Updating the strength of each dependency following a system modification only provides designers with a representation of the new 'as-is' system. This limitation has been highlighted in prior research acknowledging that: "Avalanches occur when unexpected change multipliers are encountered or when change margins of known multipliers are used up" [20] and "Loops could be included in the analysis to allow the prediction of additional changes to the initiating systems" [23]. There is a need for forecasting the ramifications of many successive independent modifications so that designers can plan for them before the system is fielded. This is especially true for systems that are expensive, long-lived, and expected to meet many new requirements.

We begin addressing this forecasting challenge by linking *excess* and change propagation probabilities. This linkage is supported by Martin and Ishii [24] who suggested that one method for reducing a component's sensitivity to change is increasing its "headroom" (i.e. adding *excess*). We accomplish this by extending an existing method, CPM, to forecast the ramifications of *excess* consumption on propagation probabilities for multiple, independent successive modifications. We start by introducing fundamental theory on change propagation and *excess* in Section 5.2, as they provide the foundation for how we model Dynamic Change Probabilities (DCP). An overview of the approach and methodology used to study DCP is described in Section 5.3. Section 5.4 is studies existing and synthetically generated systems with system level metrics, and Section 5.5 is a more detailed component level analysis of a single system from literature. Section 5.6 discusses the results overall and is followed by conclusions and limitations of the study.

5.2 Background

This chapter requires a familiarity with the concepts of excess and change propagation and a thorough understanding of CPM which this chapter extends to include DCP. Specifically,

this chapter posits that if a system undergoes multiple, independent modifications, the *excess* within the system (implicitly captured in step 3) may change. If *excess* is consumed by modifications (both directly and indirectly), then the modified components become less flexible. Less flexible components have a higher propensity to multiply change [20], which can be represented by increased change propagation likelihoods. This invalidates the originally elicited likelihoods. Change propagation likelihoods may be updated following each change but only provide guidance for the then-current system.

Therefore, we propose that simulating multiple, successive system changes can be accomplished by 1) identifying which components are modified when a change occurs using a change propagation tool, and 2) increasing change likelihoods as means of modeling the impact of consumed *excess*. The next section introduces a method for modeling these Dynamic Change Probabilities by combining *excess* consumption and CPM to explore the ramifications of DCP on system architecture after multiple, independent changes are executed.

One additional note for this chapter is that the systems in this chapter are of different scales (from component to subsystem). The practice of discretizing a system into components and evaluating the design dependencies is applicable at an arbitrary scale so for simplicity in this analysis, we refer to each element in the analyzed system as a component.

5.3 Method for Studying DCP

This section begins with an approach for unifying change propagation and *excess* consumption (with associated assumptions), as presented in the high-level pseudo-code shown in Figure 5.1. We discuss the procedure for identifying components affected by a single modification in Section 5.3.1. The process for modeling *excess* consumption by increasing change propagation likelihoods is described in Section 5.3.2. Finally, Section 5.3.3 describes

component and system metric calculation and termination criteria when analyzing change propagation.

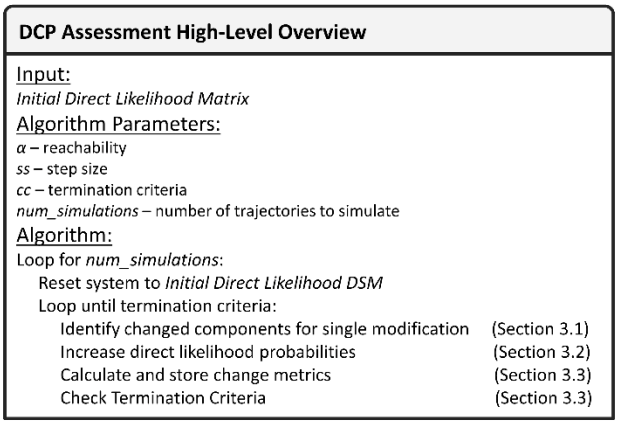


Figure 5.1: Method Pseudo-Code overview

We begin with the direct probability matrix created as part of the CPM approach. The direct probability matrix is based on a combination of component dependencies, change likelihood values defined by system experts, and calculation of the combined likelihood matrix. Modeling Dynamic Change Probabilities builds on CPM by sampling the space of potential sequence of future modifications and the impacts those changes have on the system. The direct and combined likelihood matrices from the CPM approach will be called the "initial" matrices and are used as the starting point for analysis. Building on CPM stems from a recognition that system modeling and expert elicitation requires significant effort. A firm already using CPM can apply this analysis with minimal additional effort.

By modeling Dynamic Change Probabilities, the insights of CPM are extended with forecasts of future change propagation behavior following a sequence of independent system modifications that each consume system *excess*. When a system modification is initiated, we assume that *excess* is consumed in the initiating component and in the components affected by

the change. Since *excess* is implicitly captured in the change likelihoods, we assume they increase as *excess* is consumed.

The method described in the following subsections is an exploratory study of the ramifications for the existence of DCP achieved by extending CPM. As an exploratory examination there are values assumed based on prior research and values that depend on the specific system and its context. The goal of this paper is not a defense of the specific values used but assessing the general impact of the phenomenon itself. The values assumed in this section can be modified for a particular system. For our purposes, the values chosen do not substantively change the overarching trends and conclusions.

5.3.1 Changed Components Identification

The purpose of this section is to identify all components affected directly or indirectly by modification of the initiating component, shown as pseudo-code in Figure 5.2. We begin by creating a temporary DSM for the propagation simulation and selecting an initiating component. Supporting components are then identified using the direct likelihood matrix (which is treated as a weighted adjacency matrix).

Each dependency is sampled by comparing the edge-weight against a draw from U(0,1). If the draw is smaller than the value in the cell, the component in the associated row (the j component from Figure 2.3) is added to the set of supporting components in the next change-step. Each identified supporting component is classified as being one “change-step” or “change-jump” away from its initiating component. After sampling each edge from the initiating component, all propagation likelihoods on the matrix are multiplied by a reachability factor (α). The reachability factor represents the diminishing likelihood of change propagating through multiple components as discussed in Koh et al. [45].

Next, the dependencies for each component in the change-step set are assessed to see if they propagate change. Newly identified supporting components are added to the following change-step set and the reachability adjustment process is implemented. This loop repeats until a next change-step set is empty after the entire current change-step set has been assessed.

Propagation Simulation

Input:
temp_direct_likelihood_DSM as array
Algorithm Parameters:
α – reachability
Algorithm:

```
Select initiating component (IC) from appropriate distribution
Initialize change_step_list and already_modified_list as lists containing IC
Initialize change_step_output as empty dictionary
change_step = 0
Loop:
    change_step_output[change_step] = change_step_list
    Initialize next_change_step as empty list
    For modified_component in change_step_list:
        For dependency_value in modified_component:
            If dependency_value > U(0,1):
                If component not in already_modified_list:
                    append component to already_modified_list
                    append component to next_change_step
    If next_change_step not empty:
        change_step += 1
        temp_direct_likelihood_DSM = temp_direct_likelihood_DSM * α
    else: break
```

Figure 5.2: Propagation simulation pseudocode

One assumption is that a component may only be added to a single change-step set. This assumption precludes a component being changed multiple times in the same modification as assumed in CPM. The component has already been opened for modification so any additional modifications could be performed along with the one initially required.

For the studies conducted in this paper, the initiating component is uniformly selected from the set of all components. However, expert knowledge regarding planned changes, as in Koh et al. [45], or predictions about the rate of change could be used for refining the sampling distribution. We also use a reachability value of $\alpha = 0.4$, as it limits the reachability of change propagation beyond four steps to less than 1%, again as proposed by Koh et al. [45].

5.3.2 Updating Direct Likelihood Probabilities

Having identified which components are affected by a proposed modification, we next increase the direct likelihood probabilities for those components. These probabilities are increased as a reflection of component *excess* being consumed by a change thereby decreasing the changeability of the system by increasing the likelihood of change propagation. The magnitude of the increase depends on the step-size parameter, reachability value, and change-step set of the modification.

This manner of modeling *excess* consumption does have limitations. The most significant is that component *excess* information is aggregated into a change likelihood value. *Excess* must therefore be treated as a generic property of the component. This obscures detail of component interactions but limits data collection overhead requiring no more information than CPM. A second limitation is that we do not model the increase in *excess* that occurs when buffer margin is converted into *excess*. Any conversion of buffer to *excess* or the addition of *excess* by design changes may be modeled by reducing direct likelihood probabilities manually at any point in a sample trajectory but is outside the scope of this exploratory study.

Inputs for updating the direct likelihood probabilities are the step-size parameter (ss), reachability (α), the direct likelihood matrix from the prior iteration of the algorithm, and the change-step sets from Section 5.3.1. Figure 5.3 shows the pseudocode for how the change likelihood values are increased. The new dependency value is capped at 1.0 indicating that the component has no more *excess* remaining.

Propagation Probability Modification

Input:
current_direct_likelihood_DSM from propagation simulation input
change_step_output from Propagation Simulation

Algorithm Parameters:
α – reachability
ss – step-size for dependency values

Algorithm:
Loop for each *step_number*, *component_list* in *change_step_output* :
  Loop for each *component in component_list*:
    Loop for each non-zero *old_dependency_value* of component in *current_direct_likelihood_DSM:*
      new_dependency_value = min(*old_dependency_value* + ss*α^{step_number} ,1.0)

Figure 5.3: Propagation probability modification pseudocode

The step-size parameter is how the designer represents the magnitude of consumed of *excess* by each change before accounting for reachability. As shown in Figure 5.3 the step-size is the base value by which a change increases edge weights. The range for ss is [0-1]. A step-size of 1.0 would increase edge-weights to the maximum value of 1.0 after a single modification regardless of the initial edge-weight. A value of 0 would not increase edge-weights following a modification resulting in a system with static propagation probabilities. As the step-size parameter decreases, modifications consume smaller quantities of *excess* and the components absorb more future changes. This value used should be informed by the scale of anticipated modifications for the system under examination. This requires knowledge of planned or likely modifications and how impactful those modifications are expected to be. For this analysis is set at 0.1. This represents relatively small *excess* consumption for each modification which creates sample trajectories with more values permitting higher resolution for change metric determination.

The reachability parameter is based on the concept that the further change propagates from the initiating component (the higher the change step number) the less impactful the resulting modification becomes. We assume that an affected supporting component further from the initial modification will consume less *excess* in supporting the modification. This is

analogous to the inclusion of reachability in the combined change impact value in Koh et. al. [101].

The value selected for the reachability parameter for this analysis is the same 0.4 determined by Koh et al. [101]. A lower value would limit *excess* consumption to the first few step-change sets while a higher value would increase *excess* consumption in more distant step-change sets. The exact behavior of change propagation within a single system would necessitate the selection of a reachability value appropriate for that system. We chose a value of 0.4 in this study as it demonstrates the effect that modifications increase the propensity for change propagation.

As an example of how the three components of change probability updating work, assume α=0.4 and ss=0.1. In this case the change initiator dependency values are increased by $0.1*0.4^0$=0.1. A component modified directly by the change initiator would have its dependency values increased by $0.1*0.4^1$=0.04, and a component modified in the next step would have its dependency values increased by $0.1*0.4^2$=0.016.

5.3.3 CPC Score and Sample Trajectory Evaluation

We introduce two measures of how change propagation within the system increases as *excess* is consumed. Each is calculated using the **combined** likelihood matrix. These measures form the bases of analysis for the analysis in following sections. One metric reports a component-level measure of the magnitude of that component's contribution to change propagation and another is a system-level metric for the expected impact change propagation has when making a modification. These metrics are calculated after each modification.

We start by using the updated direct likelihood matrix from Table 2.3 and calculate the corresponding combined likelihood matrix using equations 2.3-2.5. The combined likelihood

matrix is a conditional probability table where the columns are the probability of a dependent component changing due to the modification of the component in the column's heading. The sum of the column is therefore the expected number of changes that would occur if the column's component changes. We introduce a component-level measure called the *component* Changes Per Change (cCPC). A lower cCPC value indicates a component with less propensity for propagating change. These values are stored for each component each iteration.

We also introduce a system-level metric that we call the *system*-CPC score (sCPC). The sCPC is calculated by taking the mean of the *component*-CPC scores. The sCPC represents the expected number of supporting component modifications required for a random initiating component. If a non-uniform distribution is used for initiating component selection, the sCPC calculation can be weighted with this information. An example calculation is shown in Figure 5.4.

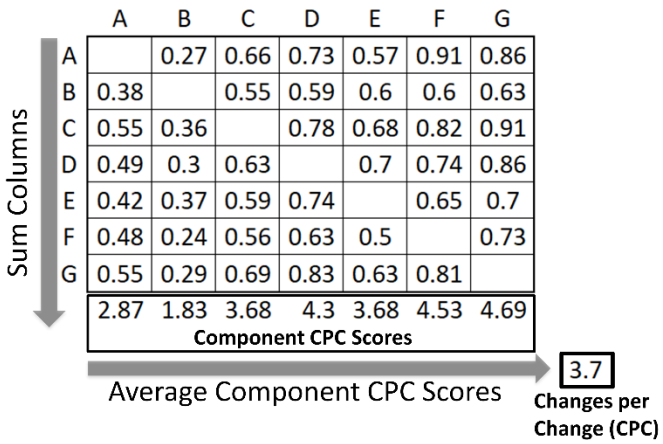


Figure 5.4: Combined likelihood matrix showing how sCPC and cCPC metrics are calculated

A terminal direct likelihood matrix occurs when each existing design dependency has reached the maximum value of 1. No remaining *excess* exists in this state. The terminal sCPC is calculated using the combined likelihood matrix associated with the terminal direct likelihood matrix. Each iteration the sCPC is checked to see if it is within a prescribed tolerance of the

terminal sCPC. If so, the trajectory is terminated; otherwise another iteration occurs as described in Section 5.3.1.

The sequence of sCPC from initial to terminal configurations is called a sample (or change) trajectory. Since the model is stochastic, each sample change trajectory varies. A system's exact change trajectory cannot be known a priori so the space of possible change trajectories must be sampled as part of the analysis. The number of sample trajectories is denoted as **n**.

A comparison of trajectories between systems (or variants of a single system) requires a measure of overall goodness. A sample trajectory sCPC scatter plot for a helicopter (referenced in Table 5.1), is shown in Figure 5.5. Here, n=100 sample trajectories, reachability ($\alpha = 0.4$), and step-size (ss = 0.1). A generation is one complete modification including change propagation and propagation probability increase (e.g. the 5^{th} generation is the fifth fully executed independent modification).

The regions of the sample trajectories used for calculating the two metrics are highlighted in Figure 5.5. The linear region (approximately the first two-thirds of the trajectories) is used for calculating the mean slope metric. A liner regression is fit to each sample trajectory in this region. The mean slope metric is the average gradient coefficient from the regressions. This measures of how quickly the system requires additional supporting modifications for each initiating modification (e.g. a value of 0.25 means each modification will on average require modification of an additional 0.25 components).

The region of termination is where sample trajectories reach the terminal state (all direct matrix dependencies have a value of 1.0). The generation value at which each sample trajectory terminates is collected and averaged to create the average max generation metric. A larger

average max generation score suggests that the system absorbs more changes indicating more changeability.

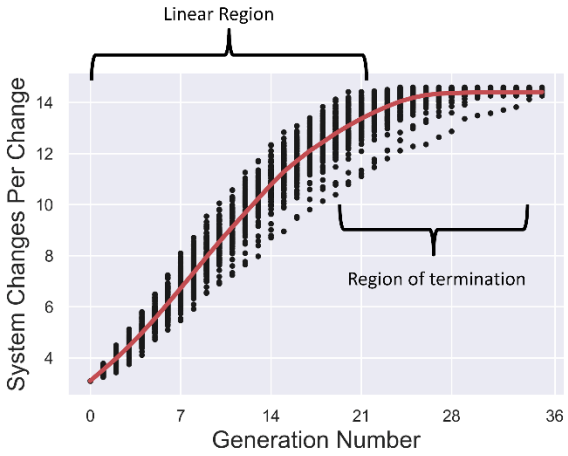


Figure 5.5: A sample plot of sCPC vs. Generation for n=100 including mean line and both metrics for Helicopter

5.4 System-Level Study of DCP and System Architecture

We have described how current change propagation methodologies do not consider multiple, independent system modifications. We have also described how modeling Dynamic Change Probabilities, based on the assumption that *excess* is consumed by a modification, can give designers insight into how the severity of change propagation might increase as additional independent modifications are executed. This section is dedicated to analysis using system-level metrics (i.e. sample trajectories consisting of sequential sCPC values) and system-level properties (e.g. the total number of high-degree components). Section 5.5 examines the contributions of specific components.

We begin this section by studying systems published in the literature. We compare sample trajectories and metrics with architectural features like number of components, number of edges, and the presence of hub components. In doing so, we demonstrate that system architecture features must be further explored so that we can understand their impact. The second half of this

section is a more rigorous analysis using synthetically generated systems. This analysis tests the hypothesis that that all else being equal, a higher number of system hubs limits the severity of DCP.

5.4.1 Study of System Architectures from the Literature

5.4.1.1 Experimental Setup

We start by considering the five systems listed in Table 5.1, whose direct likelihood matrices have been reported in the literature. Each analysis uses reachability values of 0.4 (α = 0.4) and a dependency value step-size of 0.1 (ss = 0.1). Each system is simulated with n=100 sample trajectories. Only likelihood matrices and results pertinent to the discussion are shown in the paper, while the remainder are available in the appendices.

Table 5.1: Test system properties and references

System	Number of Components	Number of Edges	Reference
Grill	6	14	[102]
Hair Dryer	6	21	[103]
Fan	9	29	[102]
UGV	14	44	[36]
Helicopter	19	110	[104]

5.4.1.2 Results

The sCPC of sample trajectories are plotted and the two metrics described in Section 5.3.3 (average changes until termination and average slope for linear region) are calculated. We first analyze the relationship between the mean slope metric and the properties of the five sample systems. Table 5.2 contains the mean slope metric and r-squared value for the resulting line. Also included is the largest difference between initial and second generation sCPC value. This is listed because it demonstrates how quickly a system could deviate from a static probabilities assumption for even a small step-size.

The key insight from the data in Table 5.2 is that sCPC increases at a higher rate in systems with more components and more edges. On average, the fan (6 components) will require 6 modifications before the sCPC value increases by one component. The helicopter (19 components) only requires 3 modifications on average. This provides further evidence of the challenge of making modifications to more complex systems. The number of component modifications required when making a single system change increases more quickly than with smaller systems. We posit that this phenomenon contributed to the design challenges when modifying the F/A-18. Each modification resulted in more necessary compensatory modifications until costs became unsustainable.

Table 5.2: DCP metrics for tested systems in order of increasing number of edges

System	Mean Slope Metric	R^2 Value	Max 2nd Gen. sCPC	Initial sCPC	Terminal sCPC
Grill	0.12	0.96	0.47	1.45	3.74
Hair Dryer	0.19	0.98	0.61	2.96	4.86
Fan	0.16	0.94	0.62	2.03	6.56
UGV	0.26	0.97	0.77	2.01	9.61
Helicopter	0.35	0.94	1.40	3.10	15.85

Sample trajectory plots of the grill and hairdryer systems are shown in Figure 5.6. We show a scatter plot of the sCPC vs generation for every sample trajectory, co-plotted with the mean sCPC value for each system. While each system has six components, the plots have two notable differences. First, the variance of sCPC calculated from the sample trajectories is smaller for the hairdryer. When we average the sCPC standard deviation from each non-endpoint generation, we find that the standard deviation of the hairdryer ($\sigma = 0.042$) is 60% of the grill ($\sigma = 0.071$). A second difference is that the grill has a larger average max generation value ($\bar{l} = 14.0$) than the hairdryer ($\bar{l} = 7.4$) and has lower initial and terminal mean sCPC values.

Figure 5.6: Sample trajectories with calculated mean for the grill and hairdryer

Explanations for these differences can be developed by examining the direct likelihood DSMs for each system, as shown in Figure 5.7. The hairdryer has significantly more edges (14 for the grill vs. 21 for the hairdryer), and more edges increase the probability that change will propagate by providing more opportunities to do so. Holding all else equal, a system with more edges reaches its terminal state in fewer generations.

Hair Dryer

	1	2	3	4	5	6
1		0.6	0.9	0.9		0.3
2	0.6			0.6	0.3	0.9
3	0.6			0.6	0.3	0.9
4	0.9	0.9	0.6			0.3
5	0.6		0.3	0.3		
6		0.3	0.3			

Grill

	1	2	3	4	5	6
1		0.5	0.2	0.5	0.3	0.9
2	0.2		0.6			0.7
3		0.4				
4	0.3				0.1	
5	0.5					
6	0.7	0.6				

Figure 5.7: Hairdryer and grill direct likelihood DSMs

Another difference is that the edges for the grill are concentrated in a single high-degree component that connects the other components. The higher variance for the grill's trajectories in Figure 5.6 is hypothesized as a result of this component. Intuitively, an increase in the grill sCPC value is highly dependent of whether component 1 is modified. In generations where component 1 is not modified, the sCPC increase remains relatively small. When component 1 is modified, the sCPC value increases sharply. By contrast, the degree distribution (mean number of edges

per component) for the hairdryer is more uniform. The size of the sCPC increase is therefore less dependent on the exact components modified due to the more homogenous degree distribution.

Sosa et al. [105] names these high-degree components “hubs” and studies their influence on system quality. Their findings indicate that systems with above-average fraction of hub components have a below-average number of defects. This is consistent with prior network analysis suggesting that systems with a power-law type degree distribution are more tolerant of random failures [106]. Theoretically the same principles apply to change propagation networks.

5.4.2 Study of Synthetically Generated System Architectures

Stronger conclusions about the hypotheses that a higher number of system hubs (i.e. a more power-law type relationship) limits the severity of DCP require many more samples than are available from literature. The time-consuming nature of data collection required for conducting CPM on actual engineered systems limits the number of available samples. We overcome this limitation by using synthetically generated system architectures so that we can extract stronger conclusions from our analysis. In this section we describe: 1) how we generate synthetic system architectures, 2) how we test the hub component hypothesis using the synthetic systems, and 3) results from our study and their implications for system designers.

5.4.2.1 Algorithm for Generating Synthetic System Architectures

In network terminology, nodes (components) are connected by directed edges (design dependencies). The degree (number of connections a component has) distribution measures the dispersal of edges among nodes. Engineered systems have in-edges (information flows in from another component) and out-edges (information flows from the component to a different component). This terminology is the basis for the discussion in this section.

Prior research [107,108] leverages advances in graph and network theory for modeling complex system behavior. In design literature, graph and network theory are more often used for modeling network resilience to failures [105,109] with some crossover into modularity [37]. Graph and network modeling are useful because the techniques provide a less subjective way for evaluating system flexibility.

This research builds on existing network theory when generating realistic synthetic test systems. Most engineered systems are poorly modeled by network generative algorithms because the distribution of edges among components is not representative. Engineered systems often have certain characteristics (e.g. a power-law degree distribution) that must be incorporated for a more faithful model of a real system. Sosa et al. [105] examined a sample of engineered systems from a quality perspective and discovered that the degree distribution of their sample was most consistent with a power-law degree distribution with cut-offs. This type of distribution is one in which a few nodes have a very high degree (many connecting edges) while most have low degree.

Building on this literature, we developed a generative algorithm for creating synthetic system architectures by modifying the preferential attachment algorithm from Barabasi and Albert [108]. The algorithm works as follows:

- First a system of components with no edges is created.
- Two uniform distributions are then created (one for the probability of an edge's tail connecting to the component and the other for the probability of the head connecting to the component).
- An edge is added to the graph by selecting the component to attach the tail from the tail distribution.

- After the edge's tail connects to a component, the probability associated with that component in the tail distribution increases. The same procedure is used for the edge's head.
- This is repeated for each edge added to the system.

The result is that most components have only a few connections while some components have many. After an edge has been attached, one parameter (γ) controls the increase in probability of attachment for a component. A large γ creates a system that tends towards a randomly generated graph. A small γ creates a system in which most edges connect to a few high degree components. The algorithm outline is shown in Figure 5.8.

Network Generative Algorithm

Initialize:

$W_n = \gamma$ for $n \in N$ ($N \equiv$ set of all components, $n \equiv$ component n, $\gamma \equiv$ initial weight, $W_n \equiv$ weight for n)

$p_n = \frac{W_n}{\sum_N W_n}$ for $n \in N$ ($p_n \equiv$ probability of attachment for n)

$P \equiv [p_0, p_1, \ldots p_N]$ (P ≡ set of all attachment probabilities)

$E = 0$ ($E \equiv$ current number of edges in system)

$E_{max} = \beta$ ($E_{max} \equiv$ max number of edges, $\beta \equiv$ user input)

Loop while $E < E_{max}$:

select n from P

add edge to n

$W_n = W_n + 1$

$E = E + 1$

update $p_n = \frac{W_n}{\sum_N W_n}$ for $n \in N$

Figure 5.8: Algorithm for network generation

5.4.2.2 Experimental Setup

Nine sets of 10-component systems are generated. Each system has 35 edges (approximately the average degree across the 5 sample systems).

- Set 1 is created with γ = 100,000 for both edge heads and tails. This generates systems where edge placement is essentially random.

- Sets 2-5 are created with $\gamma = 100{,}000$ for the edge tails, and $\gamma \in (0.01, 0.1, 1.0, 2.0, 5.0)$ for each respective set's edge head connections.
- Sets 6-9 are the same as Sets 2-5, except that tails and heads values are swapped.

Twenty test systems are generated for each combination of parameters. The initial weights of γ for each set are shown in Table 5.3. Due to the preferential nature of the algorithm the probability of a component being selected for edge attachment increases the more times the component is selected. The first and second selection probabilities in Table 5.3 provide a sample of how quickly that probability decreases with increasing initial weight. The first selection is the probability that a component will receive edge number 2 if it was selected for edge number one. The second selection is the probability that the component will receive edge number 3 if selected for both edge 1 and edge 2.

Table 5.3: Initial weight vs probability of component selection after being chosen for edge showing the influence that γ has on hub creation

Initial Weight	First Selection	Second Selection
0.01	91.8%	95.7%
0.1	55.0%	70.0%
1	18.2%	25.0%
2	14.3%	18.2%
5	11.8%	13.5%

Finally, the change likelihoods for each edge is set at 0.5. We control the initial change probabilities so that they do not confound the impact of degree distribution on system performance.

The number of hub components within the system is quantified using the method proposed by Sosa et al. [105]. This requires a hub component meet two criteria:

- the normalized degree of the component must be greater than a threshold value (0.6 for this study), and

- the number of components with degree greater than the component must be less than or equal to another threshold (4 for this study).

When normalizing the degree distribution for a component, the number of connections (either in or out) are summed and the result is divided by the total possible number of connections the component could have (N-1).

There are three metrics used in the test results. The first two are the mean slope and average maximum generations (as used in prior sections). A smaller slope value is better and a larger number of maximum generations is better. A third property is the maximum sCPC. All else being equal, a lower maximum sCPC is better because change propagates to fewer components after multiple changes.

5.4.2.3 Results

The three metrics are tabulated by the generative parameter set used when instantiating the system (In Hubs, Out Hubs, and Random), and then subdivided by how many hubs were present in each instantiation. Results are shown in Figure 5.9 and Table 5.4. This categorization allows the metrics to be compared with as-generated system architectures.

The average number of generations for each sample trajectory vs hub type and quantity is shown in the left plot of Figure 5.9. Out-degree systems have a small marginal improvement except for the 4-hub system. The results for the 4-hub out-degree system suggests that there is a non-linear relationship between the presence of hubs and the average trajectory length. The in-degree systems show a similar relationship, with 3-hub in-degree systems having marginally fewer generations and the 4 in-degree hub systems have ~8% shorter trajectories than random systems.

Similar patterns are shown in the middle and right plots of Figure 5.9. There are strong relationships between hub quantity and both mean slope and maximum sCPC improvement for out-degree systems. A system with 4 out-degree hubs is likely to have an average slope of 29% and a maximum sCPC of 44% the random system. These systems only add new supporting components at one-third the rate of a random system.

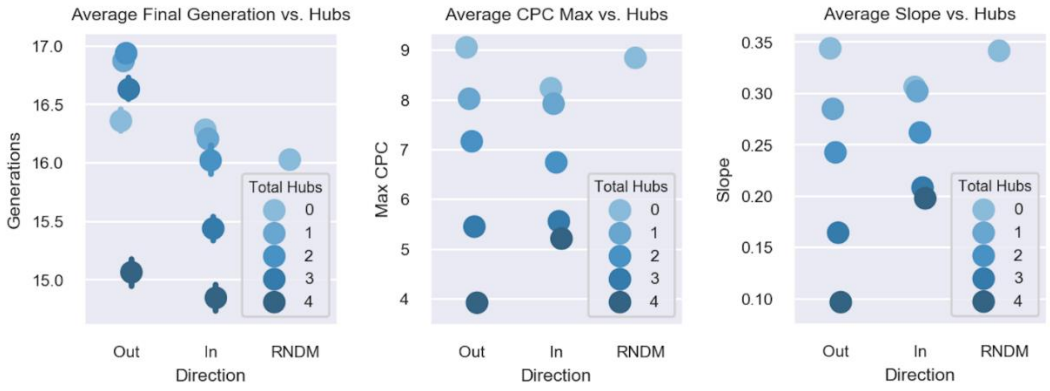


Figure 5.9: Plots showing the relationships between the number and type of hubs on three chosen properties. Plots indicate that in most cases hubs help to slow increases in propagation probabilities and reduce maximum sCPC.

In-hub systems also show improvement when compared against the random system, but the improvement is less pronounced, with both slope maximum sCPC at 59% of the random systems. The improvement in performance for each of the 4-hub systems is somewhat offset by the decrease in total number of generations. This means that systems with 4 hubs change less quickly than random systems, but also tend to reach their maximum values more quickly than random systems.

The values behind the plots in Figure 5.9 and the R^2 values associated with the mean slope metric are shown in Table 5.4. Almost all systems tested are fit well by linear regressions except for those with the most out-degree hubs. This supports the conclusion from 4.1 that hubs increase the variance of sample trajectories. Inspection of the individual sample trajectories for

these cases indicates that the cause for the deviation is that most edges in the system belong to few components. Many components in these systems have no out-edges at all causing a step increase when high out-degree components are modified. This makes the sCPC increase highly dependent on which components were changed in each generation. Generations where high-degree components are modified result in relatively large step changes in sCPC and when high-degree components are not modified there is very little increase in sCPC.

Overall, we find support for the hypothesis that a higher number of system hubs limits the severity of DCP as measured by all three metrics. The implication for designers is that hubs limit the increase in change propagation caused by the consumption of *excess* when using a uniform distribution for initiating component selection.

Table 5.4: Numerical results from architecture testing

Hub Type	Hub Number	Average Generations	Average Slope	Average R2	Average Max CPC
In	0	16.3	0.31	0.99	8.2
	1	16.2	0.30	0.99	7.9
	2	16.0	0.26	0.99	6.7
	3	15.4	0.21	0.99	5.6
	4	14.8	0.20	0.99	5.2
Out	0	16.4	0.34	0.99	9.1
	1	16.9	0.28	0.99	8.0
	2	16.9	0.24	0.98	7.2
	3	16.6	0.16	0.95	5.4
	4	15.1	0.10	0.89	3.9
RNDM	0	16.0	0.34	0.99	8.8

Now that we have a better understanding system-level effects of DCP we turn our focus to the component-level where we analyze the influence of individual components on DCP.

5.5 Component-level study of DCP and system architecture

In this section we return to the systems described in the literature (introduced in Section 5.4.1) and explore the role of individual components in Dynamic Change Propagation. We

specifically focus on how component contributions to overall change propagation increases as the change likelihoods transition between initial and terminal configurations. Gaining a better understanding of individual component contributions can inform designers about how modifications over time affect an increasing number of components for each new modification. We follow this study with an analysis regarding the efficacy of including additional *excess* in system components (lowering the initial change likelihoods) as a means of inhibiting DCP. Forecasting the potential future benefit of the extra initial investment of adding excess can help designers decide if the extra investment is worthwhile. In this section we focus our analysis on the unmanned ground vehicle (UGV).

5.5.1 Increases in Component-Level propagation between initial and terminal configurations

In section 5.4.1 we studied how the system-CPC values increase from the initial configuration throughout its lifecycle ending at the terminal configuration. In this section we expand that analysis by studying how each individual component contributes to the increasing system-CPC via its component-CPC value throughout its lifecycle. We begin by examining cCPC changes for systems from Table 5.1 to characterize general trends. We then introduce two metrics that measure how the magnitude cCPC contributions to sCPC change between initial and terminal configurations and compare these metrics with existing network property metrics.

5.5.1.1 Experimental Setup

The sample systems in Table 5.1 first undergo assessment of DCP. Each assessment uses reachability values of 0.4 ($\alpha = 0.4$) and a dependency value step-size of 0.1 (ss = 0.1). Each system is simulated with n=100 sample trajectories.

For each sample trajectory, the cCPC scores for each component are binned by generation. After all 100 sample trajectories are generated each bin is averaged. The result is a set of mean cCPC for each component, at each generation.

One additional metric used in this section is a graph theory metric called Centrality or Eigenvector Centrality [110]. This is a measure of the influence of a node in a network that captures both how important a node is and how strongly it is connected to other important nodes. The matrix form of this equation is:

$$\lambda \mathbf{x} = \mathbf{A} \cdot \mathbf{x} \tag{5.1}$$

5.5.1.2 Results

We show the Component Changes Per Change the Unmanned Ground Vehicle (UGV) at select generations in Figure 5.10. The total height of the stacked column in these figures is the terminal configuration cCPC score for each system, and the segments give perspective about how the cCPC scores increases intergenerationally. The most important comparison is between the initial (first segment) and terminal (entire bar) cCPC.

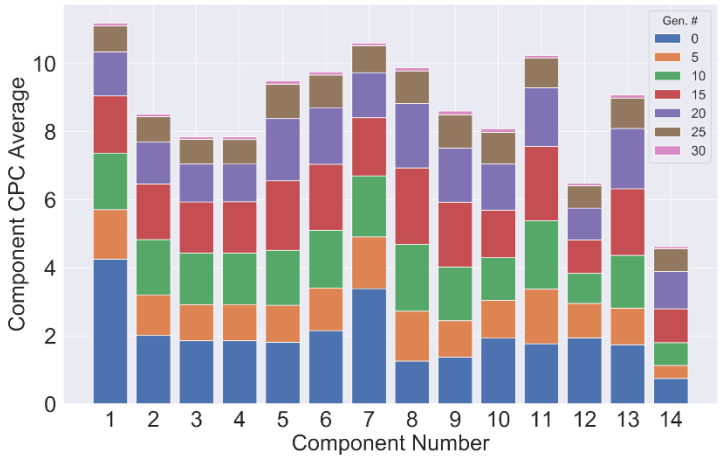


Figure 5.10: The average cCPC at various generations for the UGV showing pronounced differences between generation 0 and generation 30

For all five systems, the cCPC for all components increases. Yet, some components demonstrate much larger increases between initial and terminal configurations than others. For the fan there is higher variance in the initial cCPC scores (0.379) than in the terminal cCPC scores (0.099). By contrast, for the UGV there is an increase in variance between initial cCPC scores (0.695) and terminal cCPC scores (2.775). While the cCPC scores increase for all components (because of the change likelihood probabilities being updated as described in Section 5.3.2), they increase at different rates as governed by the underlying system architecture (the design dependencies) and initial change likelihood values.

We next seek to understand what underlying system architecture properties correlate with the magnitude of increases in cCPC. Equations 5.2 and 5.3 introduce two metrics which characterize the magnitude of increase of cCPC values. Using Equation 5.2, we calculate the change in each component's cCPC relative to total change in sCPC (initial and final values shown in Table 5.2) called the Relative Component-CPC Growth. A value of 1 indicates the cCPC increased the same proportion as the sCPC. Using Equation 5.3, we calculate the percentage change in proportion of sCPC contributed by the component between initial and terminal configurations called the Δ Component sCPC Proportion. A higher right column value indicates the component constitutes a larger proportion of the terminal sCPC than to the initial sCPC.

Relative Component-CPC Growth

$$\frac{cCPC_{\text{terminal}} - cCPC_{\text{initial}}}{sCPC_{\text{terminal}} - sCPC_{\text{initial}}} \tag{5.2}$$

Δ Component sCPC Proportion

$$\frac{\frac{cCPC_{\text{terminal}}}{sCPC_{\text{terminal}}} - \frac{cCPC_{\text{initial}}}{sCPC_{\text{initial}}}}{\frac{cCPC_{\text{initial}}}{sCPC_{\text{initial}}}} \tag{5.3}$$

Results for the UGV using these measures are shown in Table 5.5. We draw attention to these results because in Figure 5.10, Components 1 and 7 have the largest cCPC scores at the initial and terminal states. This would imply that they are the most significant contributors to change propagation. Yet, we show in Table 5.5 that the proportion of the sCPC of these component drops relative to their initial proportions.

Instead, the proportions of other components (such as 8, 9, and 11), increases more than average. Component 14 increases its proportion of sCPC by 42.1% despite growing its cCPC 58% less than average because of its relatively small initial value. These insights are important because the initial combined likelihood DSM does not indicate components 8 or 11 merit significant attention. Only after a few modifications occur, after the system is fielded, do these components become a concern when considering change propagation.

There are many factors that cause differences in cCPC growth and the change in contribution to the sCPC score. The number of design dependencies, initial dependency values, and which components are connected by those dependencies all play a role. The results of correlation testing between the two metrics calculated in Table 5.5 and other properties of the UGV are shown in Table 5.6. We conclude that none of the initial DSM properties sufficiently account for the relative cCPC growth and the percent change in sCPC contribution. Only by

calculating the percent change in the centrality between the initial and terminal DSM could a correlation be discovered that applied to both.

Table 5.5: Relative cCPC increase and percent change in sCPC contribution between initial and terminal DSMs for the UGV

Comp	Relative Comp-CPC Growth	Δ Comp sCPC Proportion (%)
1	1.03	-39.6
2	0.97	-2.6
3	0.89	-3.0
4	0.89	-3.0
5	1.14	20.7
6	1.13	4.0
7	1.07	-28.1
8	1.28	80.4
9	1.08	43.8
10	0.92	-3.7
11	1.26	33.2
12	0.68	-23.3
13	1.09	19.8
14	0.58	42.1

Table 5.6: Pearson correlation coefficients showing which network properties affect cCPC and contributions to sCPC

	Δ Component sCPC Proportion	Relative Component-CPC Growth
In-Degree	-0.36	0.53
Initial Average Weight	-0.67	0.21
Initial DSM Component CPC	-0.78	0.18
Initial DSM Centrality	-0.80	0.01
Terminal DSM Centrality	-0.19	0.67
Percent Change in Centrality	0.75	0.75

The rank-ordering of cCPCs also provides information about which components merit additional design attention. We show a box plot in Figure 5.11 where the frequency of the rank-ordered position for each component for the UGV is calculated. Lower rankings signify a larger contribution to the sCPC. Where only one horizontal line is visible, the rank for that component was almost always the same. These results support the conclusion drawn from the previous figures that components 1, 7, and 11 are the largest contributors to the sCPC. Components 6 and 8 are the next largest contributors.

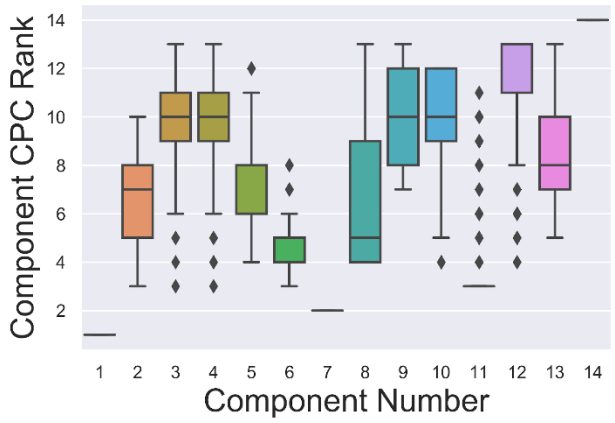


Figure 5.11: Chart showing the frequency of component rank from all generations and trials for the UGV

The wide range of ranks associated with the remaining components indicate that the specific sequence of modifications plays a large role in determining cCPC contribution to the sCPC. It can also be concluded that some components, like 12 and 14, warrant little additional design attention.

A designer can, by understanding cCPC changes and component ranks, begin forecasting the components more likely to contribute a large proportion of sCPC or experience large changes in that proportion between initial and terminal configurations. The natural follow-up question is what can be done to inhibit DCP? The architecture analyses in the system-level section suggest some approaches (e.g. reducing edges and incorporating hub components), but what can be done if the architecture is fixed? Incorporating additional *excess* to reduce initial component change likelihoods is one solution.

5.5.2 Studying DCP inhibition by Adding *Excess*

In this section we study how adding *excess* to specific components inhibits DCP. This study is performed on the UGV and is tested by adding *excess*, thereby decreasing direct change probabilities, for each component independently. The effects of adding *excess* are measured using the average max generation and mean slope metrics described in Section 5.3.3. Different

levels of *excess* are tested for each component to ascertain the strength of correlation between *excess* and the two metrics.

5.5.2.1 Experimental Setup

The test uses six levels of reduction applied to each component independently as shown in Table 5.7 for a total of 70 system configuration. Each configuration is used to generate 350 sample trajectories.

The reduction levels are applied by reducing each dependency for the selected component by a percentage. In practice the inclusion of *excess* might only affect one dependency, but because of the generic *excess* assumption described in 3.2, each edge for the selected component is reduced by the same fraction. For example, including 50% *excess* in a component with two edges of weights 0.2 and 0.8 would decrease edge weights to 0.1 and 0.4, respectively.

Table 5.7: Reductions levels applied to dependency values

Reduction Levels Used					
0%	20%	40%	59%	79%	99%

5.5.2.2 Results

The average max generations metric for each configuration is shown in Figure 5.12. Adding *excess* to components 1 and 7 has the largest impact on average max generations metric. These components are the largest contributors to sCPC so a reduction in their dependency values increases the metric. The next tier of components is: 5, 6, 11, and 13. These all show clear response relationships between the size of the reduction and the increase in maximum generations.

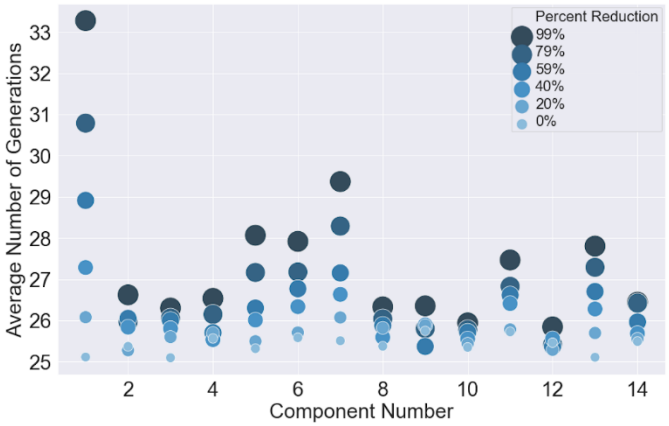


Figure 5.12: *Excess* vs Average Generations (higher is better)

Figure 5.13 shows how the mean slope metric is affected by each configuration. Components 1 and 7 again show the strongest response to the inclusion of *excess*. While the response of other components is less clear than Figure 5.12, Table 5.8 makes the correlation clearer by calculating the Pearson correlation coefficient between each metric and the 6 levels for each component. Components 5, 6, 11, and 13 form the second tier after components 1 and 7. While the correlations with the maximum average generations metric are stronger, the two metrics reveal the same pattern.

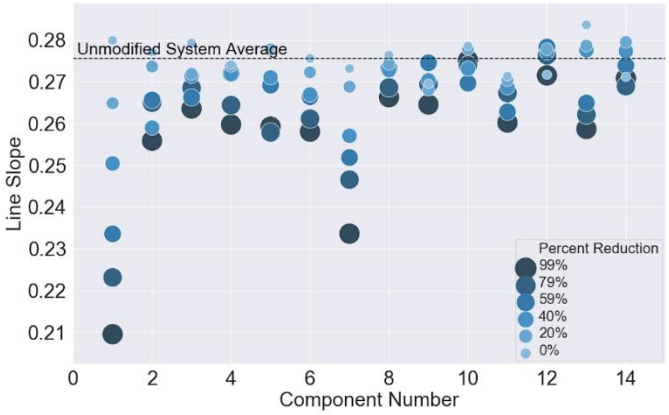


Figure 5.13: *Excess* vs Slope (lower is better) showing improvement for most components and significant improvement in some.

Comparing the components with the strongest correlations from Table 5.8 with the initial cCPC (the bottom bar from Figure 5.10), we see that some components score highly in both (components 1 and 7), but other components become significantly more important only after multiple system modifications (components 5, 11, and 13). The components with the strongest correlations offer the best opportunity for inhibiting change propagation.

In summary, this analysis shows component-level *excess* addition can be leveraged to inhibit the undesirable consequences of change propagation. For the UGV specifically, components 1 and 7 inhibit change propagation the most. This analysis provides additional alternatives in components 5, 11, and 13 that could be considered when allocating *excess* if it is not cost effective to add excess to components 1 and 7.

Table 5.8: Component-wise Pearson correlation coefficients of metric vs *excess* addition for the UGV

Comp	Max Generations	Slope
1	0.79	-0.55
2	0.18	-0.08
3	0.17	-0.08
4	0.13	-0.02
5	0.38	-0.25
6	0.34	-0.20
7	0.45	-0.21
8	0.11	-0.07
9	0.05	0.02
10	0.09	0.01
11	0.24	-0.15
12	0.04	0.05
13	0.37	-0.27
14	0.16	-0.13

5.6 Discussion

The principle contribution of this paper is a refined understanding of how, after a system undergoes successive independent modifications, change propagation shifts at both the system

and component levels. Prior research has studied the effects of change propagation in the initial system configuration. This work extends that research by exploring the effects system architecture and *excess* have on change propagation. The result is a forecast that can help predict which components have a larger impact on change propagation as the system lifecycle progresses.

The first analysis in Section 5.4.1 analyzed DCP in five systems from existing literature. A comparison of mean slope metrics suggests that sCPC increases are influenced by both the initial change propagation likelihoods and the system architecture. The average sample trajectory for a system is linear for approximately the first two-thirds of its life before leveling off at the maximum value dictated by the system architecture alone. The presence of hub components was observed to influence the variance of sample trajectories suggesting that the component selection sequence plays a larger role in how quickly DCP worsens change propagation. Finally, it was found that more densely connected systems have steeper sCPC trajectories. The steeper sCPC trajectory means that the number of supporting modifications required for each new modification is likely to increase quickly making system modifications more costly.

The analysis in Section 5.4.2 tested the hypotheses that system hubs may inhibit DCP, as suggested by the first analysis, in a more rigorous manner by generating synthetic systems in which the number of hubs was varied. Analysis of the results found that both out-degree and in-degree hubs were generally associated with an improvement in DCP metrics meaning the increase of change propagation over time was slowed. The out-degree hubs improved metrics more robustly than in-degree hubs. The engineering implication of this finding is that system hubs help limit the increases in change propagation that occur as a result of *excess* consumption.

Section 5.5.1 studied how individual components contribute to the overall increase in change propagation. Relative component contributions to sCPC were shown to vary between the initial and final configurations. This suggests that some system have stable component contributions to change propagation over time while others experience shifts in their contributions. The components of the fan were shown to have a largely similar proportion of contribution to the initial and final systems while the UGV was found to have components that dramatically shifted relative contributions. Similarly, a comparison of the frequency of a component's rank in terms of its contribution to sCPC found that some components have a very consistent rank (usually those with large or small contributions to sCPC). Other components have ranks that shift depending on the specific trajectory of the sample. Understanding which components may make large shifts in their contribution to cCPC could help designers forecast which components are more likely to shift from change absorbers to change multipliers not evident with CPM alone.

Most network properties tested, except for change in eigenvector centrality, correlated only with either the relative increase in cCPC or the change in that component's contribution to sCPC, but not both. This suggests that change in eigenvector centrality is a reasonable measure for how strongly an individual component affects change propagation throughout a system's lifecycle.

The final analysis in Section 5.5.2 studied the compensatory strategy of including additional *excess* in UGV components to inhibit the effects of DCP. The effect of additional *excess* (with corresponding dependency value reductions) for each component was tested independently. The results showed some components have stronger correlations between the inclusion of *excess* and improved DCP metrics than others. While some components with

stronger correlations were readily identifiable from the initial system CPM others were not and required further analysis for identification. Overall, the fourth analysis suggests that targeted *excess* placement can be used to inhibit DCP and may be worth including depending the ease of incorporation.

The analysis conducted in this chapter is intended as a low-overhead method for modeling lifelong change propagation for use alongside CPM. This is satisfactory for high-level analysis, but limitations exist prevent use for more detailed analysis and guidance. Prior research has demonstrated that new change propagation pathways are created as system *excess* is consumed. For example, research into the F/A-18 discussed in the background revealed that internal system volume was depleted by component additions. It therefore became necessary to miniaturize existing components only due to a lack of *excess* and not through any direct design dependency. This suggests that change may propagate both along design dependencies between components and system level attributes (supporting research by Sosa et al. [37]). This would provide additional insight into how much *excess* should be included in the preliminary system.

Most significant are the aggregation of different types of excess into one generic quantity that affects each of the component's dependencies and exclusion of costs altogether. Removing these limitations would allow for detailed system modeling that provides specific guidance on the placement and forms of excess most useful to limit the worsening of change propagation. A future improvement would link the details of the modifications with the degree of excess consumed when updating change probabilities. Chapter 6 builds on this chapter by relaxing these limitations (most significantly the "generic excess" assumption). Chapter 6 also incorporates model enhancements like component costs, system specific connections between components, and change drivers with the tradeoff of requiring a more detailed and complex system model.

Chapter 6: Excess Based Change Propagation

Prior research suggests that excess can limit change propagation and reduce system modifications. Reducing change costs increases system flexibility, permitting adaptions that satisfy uncertain future requirements. The benefits of excess, however, must be traded against higher costs of the initial system and likely performance decreases. Assessing the benefits and costs of excess requires evaluating what forms, locations, and magnitudes of excess inclusion are optimal. This chapter improves the state-of-the-art in two ways. First, prior research has generally assessed excess in system-level properties (aggregating component properties into a single metric). The approach presented in this chapter extends excess assessment to the component-level so that the effects of excess on change propagation may be explicitly captured. Second, this approach holistically assesses the value of excess by evaluating both its costs and benefits. The approach borrows from Decision-Based Design and Model Based System Engineering (MBSE) in creating a generic modeling method capable of excess valuation. A desktop computer example is used for demonstrating how excess is valued in a system and the potential gains associated with excess inclusion when mining cryptocurrency. A single component optimization of the power supply capacity for the desktop is assessed to be 750W which balances initial cost against the future flexibility. A system level optimization then demonstrates identification of critical change propagation pathways and illuminates both where and how excess may be included to inhibit change propagation.

6.1 Introduction

This research by motivated by asserting that all engineered systems are designed in a specific context comprised of 1) requirements and performance goals associated with customer needs and preferences, 2) technologies and tools available to the designer, and 3) consideration

of the other systems that will interface with it. Modern systems also require a significant amount of effort (time and money) in their design and construction, a process that occurs in a rapidly evolving context with an increasingly frenetic pace of change [15].

Lengthy system lifecycles are desirable for maximizing the benefit derived from complex systems, but rapid requirement change diminishes the rate of return as the system ages. Designers must decide how uncertain future requirements should be addressed. Barber et al. [111] describes three strategies:

1. Design to current requirements only
2. Design to meet predicted end-of-life requirements
3. Design so that the system may be modified to adapt to new circumstances

The benefits of each strategy must be evaluated by considering improvement in overall lifetime value. Designing only for current requirements is effective when the system can be inexpensively replaced (e.g. simple consumer electronics). If a significant change in context does occur, the system is replaced with an updated model.

Designing for end-of-life requirements is effective when there are prohibitive system modification costs. Designers must then preemptively plan for hypothetical context changes. Saleh et al. [112–115] developed this type of analysis for telecommunications satellites. Uncertainties in costs and value generation were holistically modeled by combining cost estimating relationships, obsolescence modeling, and an explicit value generation model. By considering the uncertainty associated with system costs and value generation, Saleh et al. demonstrate how design decisions can be informed about a satellite's optimal lifespan and number of transponders.

Embedding changeability requires assessing whether uncertain future benefit warrants the definite costs associated with that changeability. This resembles designing for end-of-life requirements where the costs of designing and building the system are used in conjunction with a value generation model for optimizing system lifecycle value. However, there is the increased difficulty of accounting for system modifications (cost and benefits). Adding estimates for modification costs, and how they can be reduced with embedded changeability, is far from trivial as ontological debates still occur about the very definition of changeability [4,31,116,117]. Saleh et al. [4] compares the state of changeability research to safety research decades ago as "vague and difficult to improve, yet critical to competitiveness".

It has also been shown that an absence of *excess* within a system can result in unforeseen change costs or even premature system obsolescence [20]. However, the inclusion of *excess* incurs additional initial costs. Therefore, engineers must make decisions about whether *excess* should be included, and if so, in which components, and to what degree. Existing design tools do not adequately answer these questions.

In this chapter the Decision Based Design (DBD) framework developed by Hazelrigg [28] is used as the foundation for a value generation modeling approach with particular interest in the influence that component capability and interface definitions – fundamental concepts associated with *excess* – have on system lifecycle performance and the propagation of system modifications. The DBD framework requires a system representation (including components, interfaces, and system performance determination), a process for modeling change propagation, and a value generation model. This chapter demonstrate how a Model Based System Engineering (MBSE) implementation can be used for linking these model elements and managing model complexity.

Specifically, this chapter: 1) uses the DBD framework for evaluating system changes, change propagation, and value of *excess*, 2) describes how this analysis can be implemented using an MBSE-based representation, and 3) demonstrates how this analysis gives insight about optimal component selection and *excess* inclusion for a desktop computer test-case using historical data. The following section reviews the existing research necessary to be combined and/or extended to support this effort.

6.2 Background

This section assumes basic knowledge of Excess and CPM as covered in Chapter 2. One additional related concept (Real Options) and one additional modeling approach (Model-Based Systems Engineering) are necessary and are included below.

6.2.1 Real Options Analysis

Real options analysis was derived from a method used to estimate value of financial options under uncertainty in financial markets. A financial option is the right, but not the obligation, to buy or sell shares for a fixed price at some point in the future [118]. The appeal of an option is the hedge it provides against uncertainty. By purchasing an option, the holder spends money now to protect against future uncertainty. Real options analysis uses similar methods with the aim of assessing the value of options on non-financial (or real) investments. Engineering design researchers have advanced the idea of real options to include the value of real options "in" a system. Real options in a system are options that provide flexibility though some attribute or design feature. This could be preparing the foundation of a bridge during initial construction to add a second deck later if needed [119] or a parking garage with initial construction made to support the addition of more levels [120].

The value of real options analysis is that is encourages designers to consider alternatives to a point design. This includes traditional options like the option to delay a project or the option to stage project growth. Real options can also be expanded for considering the impact system margin has on the future system value in the presence of uncertainty, a part of this research.

6.2.2 Model-Based Systems Engineering

Finally, this research takes advantage modeling approaches like Model Based System Engineering (MBSE). The premise is that instead of documents as the paper of record for a system, the design is instead stored digitally in a series of interconnected objects. These objects can represent system connections, specific analysis, requirement capture, and any other aspect of design. The interconnected nature means that it is easier to track what impact one block may have on other in the system. Delligatti [121] provides an introduction to a popular language for implementing MBSE called SysML. This research incorporates the notion that different aspect of a system can be modeled as distinct objects connected to one-another to form a complete system which may then be used as a basis for analysis.

6.2.3 What is missing and what can we learn?

Many open questions remain about evaluating the value of system changeability. Prior research has focused on system-level metrics that measure architecture modularity. We view *excess* as a promising new avenue for increasing system net lifetime value by making the system more changeable, but no satisfactory analysis is currently available for evaluating the expected return on excess embedded at the component-level. Therefore, there are no frameworks for assessing where component-level excess should be included, to what degree it should be included, or even if excess is worth including at all.

Prior research does, however, provide guidance about what phenomenon should be included in an evaluation. First, evidence suggests that excess can make a system both more robust (insensitive to changing requirements) and more flexible by inhibiting change propagation and reducing the effort associated with system changes. Linking these two effects with system costs and value generation is the key missing step.

We explore the value of excess in this paper by considering the following design scenario: at the beginning of 2011 an individual takes part in the cryptocurrency rush. They generate value by building a desktop computer dedicated solely to cryptocurrency mining. The individual will reassess the computer's performance at the beginning of 2014 and make a decision about whether the system should be upgraded, left as-is, or retired and a new system purchased.

After the decision has been made, the simulation proceeds for another three years. To the extent possible, the components (performance parameters and costs) used are those found in historical records. Examples of component specifications and initial system builds are included in Section 6.7.

Some assumptions are made regarding the measurement of system performance and the specifics of how cryptocurrency is earned. Computational performance is measured by the number of Floating-Point Operations per Second (FLOPS) a system can generate. It is also assumed that there is a direct conversion between the total number of Floating-Point operations performed and the currency earned.

We perform two analyses using the example. The first analysis involves lifecycle optimization using different electricity costs and currency conversion rates. We then assess how initial power supply capacity impacts lifecycle value. By explicitly accounting for system

modifications and the resultant change propagation, we assess system lifecycle value in a manner not possible using existing methods in the literature.

6.3 Overview of The Approach

Our goal is evaluating the benefits of component-level *excess* by assessing a system's net lifetime value. This process is challenging because a lifecycle simulation for a changeable system requires many features. The model must be capable of representing how:

- system modifications are made
- changes propagate within the system
- system value is affected by modifications, including incurred change costs and a new value generation rate

We adopt a modified form of the DBD framework, as shown in Figure 6.1, as a process guide. The DBD framework is embodied by a system model, a model of system change propagation, and a discrete time simulation.

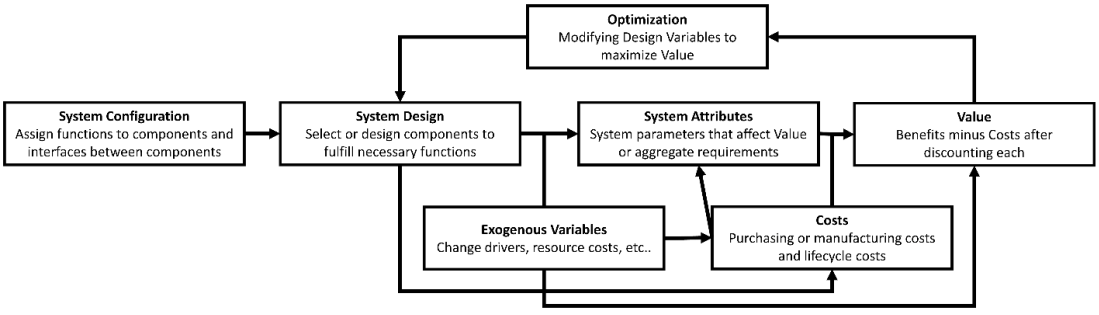


Figure 6.1: Modified DBD framework adapted from Hazelrigg [28] adopting the perspective of the system operator. Value is used instead of utility and the demand box is dropped.

The main focus of this effort is the system model, as described in Section 6.4. The key insight of this work is that embedding a computational network within a SysML-like block structure creates a flexible system representation. The model has two key features: 1)

automatically assessing whether the system components satisfy both internal and external constraints and requirements, and 2) algorithmically updating system attributes based on component attributes and exogenous variables. The first feature allows feasibility testing of different configuration combinations. The second feature allows for the assessment of lifecycle value by simulating scenarios involving arbitrary time-dependent exogenous variables.

The lifecycle simulation is a discrete time model that tabulates system costs and value over time with time-dependent exogenous variables (e.g. the price of electricity). Additionally, predefined strategies for initiating system change can be evaluated for identifying which components provide the best overall value. This simulation is described in Section 6.5.

The change process described in Section 6.6 is an automated method for generating updated system configurations by adding or replacing components. The system model is used for identifying new configurations allowed within existing constraints and requirements. This process also permits the tracing of violated requirements and creates a list of change options (changes to other components) that correct the violation.

6.4 Creating object-oriented models

The system model system representation capturing aspects of the system pertinent for determining system performance or change propagation. Conceptually, the system model is composed of system components and the flows connecting them to requirements, other components, and external objects (e.g. electricity from an outlet). The implementation for the system model is a computational network embedded within component "blocks" (similar to SysML blocks) that are connected via virtual interfaces. This section describes the process by which the system model is generated, as shown in Figure 6.2, where groups of activities are grouped by steps of DBD framework.

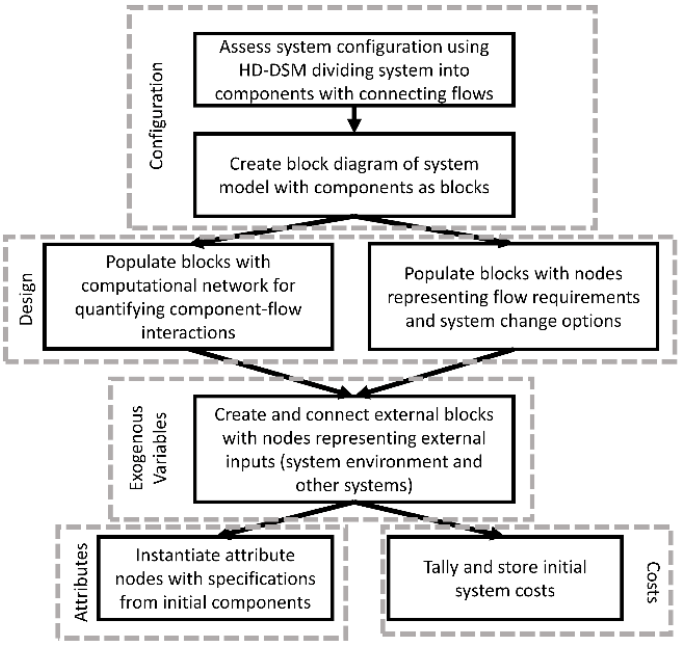


Figure 6.2: System model framework describing the steps and sequencing for how the system model is assembled. The dashed boxes indicate where DBD boxes are embodied

6.4.1 System Configuration

Modeling a system requires assigning functional flows to components and assigning connections between components. This is the ***system configuration*** block in the DBD framework, where the focus is only on how components are organized and connected. Constraints and requirements are also specified at this stage, however, specific values are not assigned. We build on the work of Cansler et al. [52] and White and Ferguson [53] by modeling a system as a set of object-oriented programing (OOP) “objects”. Rule-based procedures are also established that define how components operate and interact. This also establishes the system flows along which change may travel, providing the groundwork for change propagation analysis.

The work done by Tilstra et al. [122] developing the High-Definition Design Structure Matrix (HD-DSM) provides a method for representing the initial system configuration –

components and their connections - across multiple dimensions. We connect components on the dimensions of the flow types described in Allen et al. [50], without defining flow quantities.

6.4.2 System Design

Next, we specify mathematical models of component capabilities that formally capture how flows are generated, modified, and consumed. We accomplish this using computational networks embedded in component objects (or blocks). Flows are subject to constraints and requirements from basic physics (e.g. energy must be conserved), interactions between components (e.g. interfaces must conform to connected components), and external requirements. The resulting component blocks may then be added, removed, or replaced with a rules-based procedure. This provides the means for automating change propagation. The following sub-sections describe how these component blocks are generated. Using MBSE with embedded computational graphs for automating system modifications is an original contribution of this research.

6.4.2.1 Flow Tracking via Computation Network

Quantitative flow modeling requires algorithmic manipulation of flows so that changes are appropriately propagated. We propose modeling flows using computational graphs. Computational graphs are commonly used in Deep Learning Artificial Neural Networks for simplifying the backpropagation required with gradient ascent.

A computational graph, as shown in Figure 6.3, is composed of two parts: nodes and edges. Each node contains either a value assignment or a computation. The edges connecting the nodes specify how values are passed. For example, if the values presented in Figure 6.3 represent electric power, then blocks *a*, *b*, and *d* consume the power provided by node *e*. Node *e* is a

calculation with directions for passing values shown by directed edges. The total power for the system can be quickly and automatically calculated by updating the computational network.

The network of nodes and edges representing component functionality are aggregated within a block that represents that component. The edges pass flow values (mass, heat, electrical power, etc.), while the nodes specify how these flows are added, removed, modified, or transmitted.

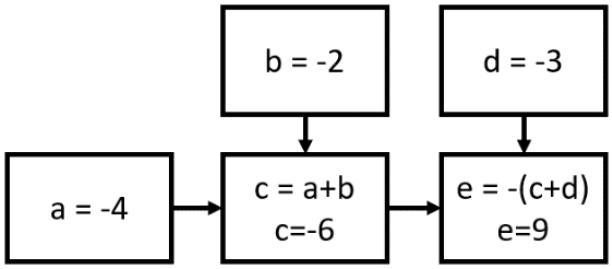


Figure 6.3: Computation graph showing how values (flows) transfer via edges.

As a demonstration, energy consumption by components connected with the power supply is modeled in Figure 6.4. The grey blocks are *calculation nodes* that take in input, perform a calculation, and return an output. The white box is an *attribute* node for the power supply and is static. The total power required by the power supply is determined solely via flows from other components and the computational network embedded within the power supply. The computational network within each component is a convenient representation for how component parameters (design variables) affect system attributes.

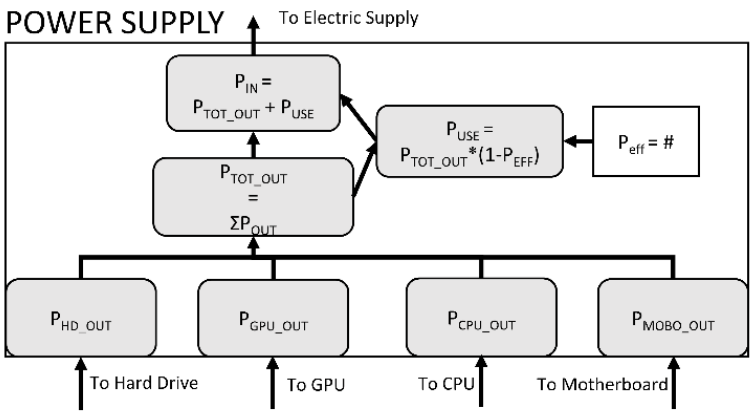


Figure 6.4: Computational graph with attributes and calculations imbedded in a block

Functions of arbitrary complexity may be contained within a node. This allows the use of more advanced computational models like finite element analysis packages. Once system parameters are calculated by a computational network the next step is ensuring the system operates within an allowable state by incorporating constraints and requirements.

6.4.2.2 Requirements Tracking

Requirements and constraints are represented by computational nodes. These nodes use flow values as input and return a Boolean value indicating requirement/constraint satisfaction. Requirement violations are returned as False and indicate that an aspect of the system is in a non-allowable state. A key advantage of the computational network is that edges provide a means of tracing backward along dependencies to uncover what modifications may return the system to an allowable state.

Building on the example from Figure 6.4, the addition of five requirement blocks for the Power Supply Model is shown in Figure 6.5. Requirement blocks have dashed edges. In this example, they are used for comparing the power output of each interfacing component against the maximum allowable values for the power supply. The power used by interfacing components is then sent to the power supply's calculation nodes. This calculation updates the total needed

power. Satisfaction of each requirement is based on the flow value and the maximum specified values for the component.

One complication is that flows may be traced back an arbitrary distance. This results in large sets of potential modifications when remedying requirement violations. This mirrors reality when a designer has many different options of change propagation pathways. This issue will be revisited in the discussion of the simulation in Sections 6.5 and 6.6.

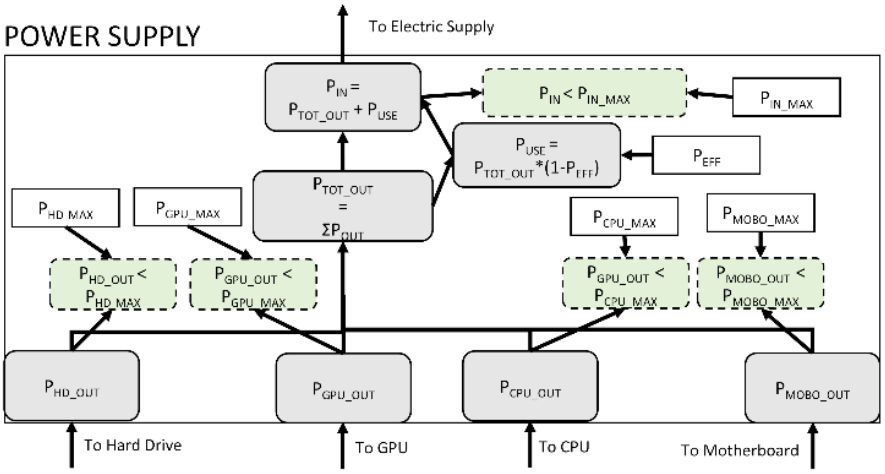


Figure 6.5: Block graph showing how requirements are implemented in the computational graph.

6.4.3 Exogenous Variables

Exogenous variables exist outside the system and represent the system's context. They impact how the system operates (e.g. the ambient temperature), its operating costs, and its generation of value. Exogenous variables are modeled as nodes in the computational network (like system capabilities) outside the system boundary. We consider two types of exogenous factors in this work, as described in Sections 6.4.3.1 and 6.4.3.2.

6.4.3.1 Technology Limits

The first impact of technology is in the specification of necessary connections between components. The functions fulfilled by each component and the flows required are influenced by the technologies used. The inclusion of specific technologies can make two systems distinct and

incompatible even though they perform similar functions. This impact is most significant when standard interfaces change. One example is a change in specifications for a standard interface (like the change in the socket for a CPU).

The second impact is on available component performance parameter sets. For example, when selecting a computer graphics card (GPU) there are sets of attributes (computational power, memory cache size, memory cache speed, power use, heat generation, cost, etc.) available. Selecting a GPU requires specifying a set of performance attributes allowed within the tradespace constrained by currently available technology. The optimal allowable attribute set falls somewhere along a Pareto frontier on which improving one attribute can only be accomplished at the detriment of another. Having the proper type of *excess* can enable computer upgrades of more advanced components without incurring excessive change costs.

For example, over time the maximum theoretical computational power for a GPU has improved as electric power consumption has fallen. There are also new graphics cards that use far less power, designed for mobile computing, that are largely similar in price to the more powerful desktop graphics cards but provide less computing power. We show the performance of three equivalent graphics cards from different points in time in Figure 6.6. The improvement of each attribute has been normalized to the 2010 value with costs adjusted for inflation. Holding costs roughly constant, the power required decreases while the computational power increases. A designer can only select from the Pareto frontier that exists during initial design, but as time passes the frontier also shifts. These shifts may be capitalized on by a well-designed system.

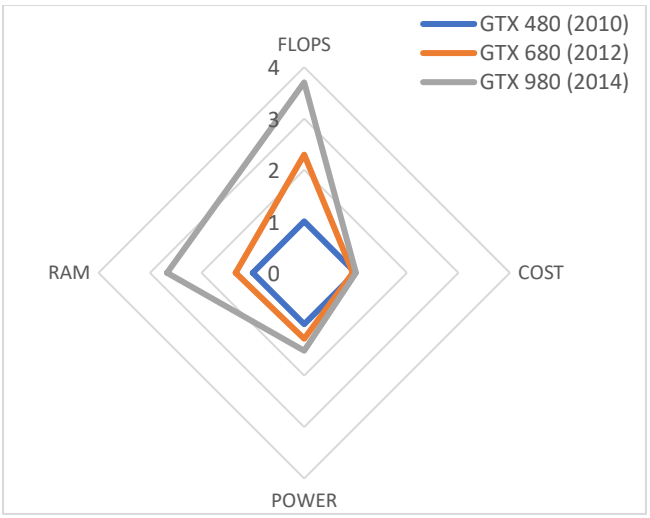


Figure 6.6: Normalized NVIDIA GPU Pareto frontier comparison showing improvement in component attributes normalized to 2010 model.

6.4.3.2 Environmental and Systems of Systems Variables

Systems exist in, and must interface with, their environment and other systems they come into contact with. Two ways that external variables are included in the system model are shown in Figure 6.7. The first is shown by the interface between the power supply and external electrical system. Each interface is labeled with a reference type. The interface between the power cord and the external power source is NEMA 1-15 Grounded. This interface standard has a maximum power rating that is accounted for by a requirement inside the interface. Specifying standard interface types simplifies interface representations.

The second interface is shown between the external ambient air and the power supply. Ambient air acts as the ultimate heat sink for computer-generated heat. The cooler the outside air, the less work required for keeping operating temperatures below their maximum. Since this interface does not require a specific interface, the flow directly crosses the component boundary.

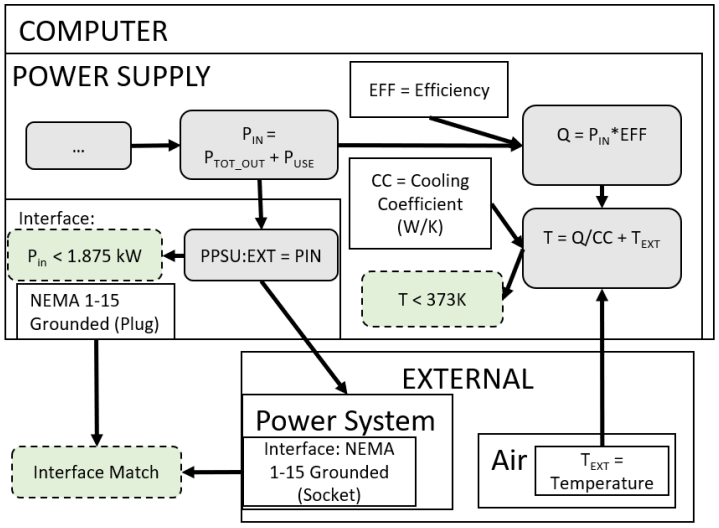


Figure 6.7: System model demonstrating how standard interface matching abstracts geometry detail and how attributes of the external system included.

6.4.4 System Attributes, Costs, and Value

System value is determined by costs (including modification and change propagation) and generated value (updated with exogenous variables and system changes). Both depend on the system attributes. Lifecycle value provides a standard metric for comparing systems and enables quantitative optimization of system configuration and design choices.

A significant challenge during model creation is that a metric for system value must be developed so that optimization can be performed. It is assumed that a designer can adequately capture and model this value. The case-study used in this research converts computational power directly into dollars via simulated crypto-currency mining for simplicity.

The costs of a system include both those associated with the initial system (design and build) and those accrued during operation. Decisions made during configuration and design influence these properties, but they are also subject to the uncertainties imposed by the exogenous variables. As described in Hazelrigg [40], the costs for a system may broken into two categories: recurring and non-recurring. For this work, recurring costs are associated with the

system's operation while non-recurring costs are incurred during system design/production, modifications, and at end-of-life.

6.4.4.1 Recurring Costs

Recurring costs and benefits are generated during system operational periods and accrue over time. This includes the cost of purchasing materials and energies required for system operation. These are assessed by modeling system attributes in a time-dependent simulation. The system attributes (e.g. electricity consumption) and environmental variables (e.g. electricity cost) are tabulated during the inter-epoch period at discrete intervals. Within each interval the system design and external variables that connect with nodes inside the system boundary are held constant.

6.4.4.2 Non-Recurring Costs

Non-recurring costs are incurred during initial system development, at the end of system life, and between epochs. Non-recurring costs are all costs associated with initial system design and build, system modification, and system retirement.

In this study, the focus for non-recurring costs are those associated with system modifications described in Section 6.6. Procedurally generating modification costs is necessary for cost determination when making system changes. This is necessary when simulating many sample system lifecycle trajectories.

6.5 Discrete-Time Simulation

Once generated, the system model can be exercised in a battery of discrete-time simulations for assessing system lifecycle value under a variety of upgrade strategies and exogenous variable scenarios. This is a key capability, as it allows explicit testing of system

changeability in a generic way by sampling possible system lifecycles and comparing the improvement afforded by *excess* against its costs.

The simulation's overview is shown in Figure 6.8 and may be conceptualized as a series of epochs as introduced by Ross and Rhodes [123]. Each epoch is a discrete period during which the system is substantially unchanged and only recurring costs are tabulated. When a modification is triggered the current epoch ends, a change is made to the system (including associated change propagation), the model's computational network is updated, associated non-recurring costs are tabulated, and the next epoch begins.

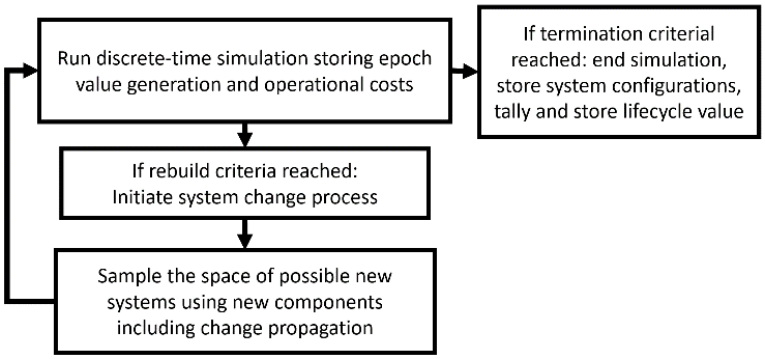


Figure 6.8: A diagram showing the simulation as divided into epochs separated by rebuilds until termination criteria are reached

6.6 Procedure for Modeling System Change

Change occurs when a system modification is made that results in violated requirements. These violations occur at requirement nodes and are identified when the computational network in updated. The procedure described in Figure 6.9 is used for making changes and addressing unsatisfied requirements.

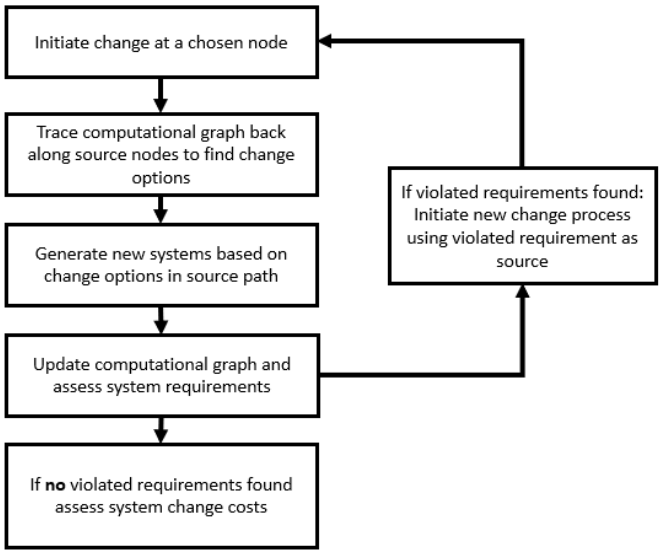


Figure 6.9: Simulation strategy for assessing change propagation and linking it to lifecycle value

6.6.1 Initiate Change at Chosen Node

The element of the model being modified becomes the source of subsequent changes. This can be any element that exists in the system in the form of a node, including requirements, system attributes, or components.

Consider an example where a desktop computer is used for the real-time control of a robotic system. Suppose a new control algorithm is implemented that requires more FLOPS. A diagram containing the pertinent system elements for this change is shown in Figure 6.10. The Iterations per Task attribute must increase because of the new algorithm's increased task time. This causes a violation of the Task Time requirement, which in turn becomes the initiating node.

6.6.2 Generate Change Pathways

After a change initiator is identified, the set of changes needed for returning the system to an allowable state must be determined. A new object type in the model is introduced representing where and how system changes may be made. These "Change Options" are represented as ovals in Figure 6.10.

Change Options nodes contain pre-written algorithms for how the exercised option changes the system. Beginning with the change initiator, the system graph is traced backwards along edges (dependencies) with a breadth-first search, creating a set of candidate Change Options. This includes options like adding or replacing components (changing the GPU or adding a second GPU to the system). Two Change Options, “Add GPU” and “Replace GPU”, are shown in Figure 6.10.

Once all applicable Change Options are identified, we assess each possible combination of change options. Newly generated configurations are then checked for violated requirements by updating the computational network. If any are identified, the violated requirement node is used as the initiating change node for a new change propagation process. Components or exercised options that were selected in a prior iteration within a change process may not be included in subsequent change propagation processes. If any violated requirements exist and all Change Options have been exhausted, the configuration is discarded.

The result is a list of viable systems that improves the change initiating node and has no violated requirements.

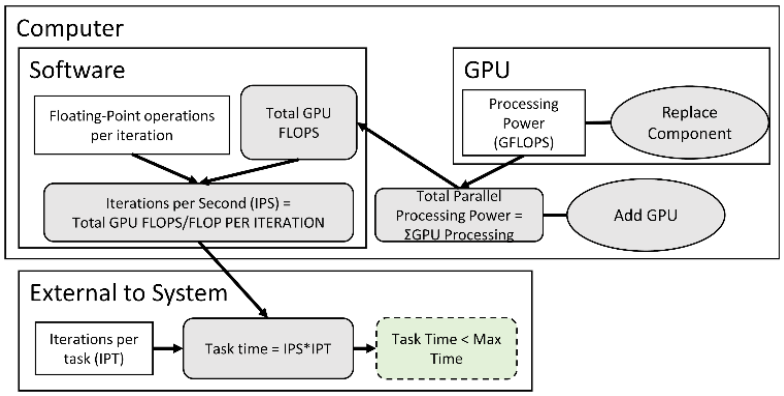


Figure 6.10: Change option example showing the “Add GPU” and “Replace Component” change options

6.6.3 Evaluation of Change Costs

Modifications incur component acquisition costs and the labor cost of assembling and disassembling the system. Each cost is procedurally generated using costs linked to associated component and its interfaces. Associating costs with components and their interfaces provides an efficient way for assessing arbitrary system changes.

Component acquisition costs are incurred when new components are purchased or built. For example, a computer modification might require a new CPU and power supply. The all-in costs of each component are included in the component's attributes.

Modifications also require the system be disassembled, old components removed, new components installed, and the new system reassembled. Each step requires that interfaces either be disconnected or reconnected. Disconnecting and reconnecting interfaces incurs effort based on the time required. Interface connect/disconnect costs are included as attributes of the interfaces included on each component.

Assessing the total cost incurred by a modification is accomplished by comparing the initial configuration with the final configuration following a modification. The component costs for new components are added and added to the costs associated with all interface disconnections and reconnections for modified components.

6.6.4 System Performance

The final task is updating the system's computational network so that values from new component attributes can be propagated to system level attributes used by the discrete-time simulation.

6.7 Desktop computer example study

We now return to the example described in Section 6.2.3 of a desktop computer generating value by mining cryptocurrency. Value generation via cryptocurrency permits a simple conversion of computational power to dollars. Additionally, historical records of computer component performance, costs, and recommended component sets exist that allow for a faithful recreation of historical technology shifts.

This section is composed of two simulations using the same system model. The first simulation assess the impact of the initial system power supply on lifecycle value, holding all other initial components in common. The second assesses the lifecycle value of three tiers of systems (Entry Level, Mid-Range, and Dream) suggested by an industry publication. These examples illustrate how the system model and exogenous variable scenarios can be used for evaluating long-term system performance and the holistic assessment of initial system design choices.

6.7.1 Common System Model

A common system model for the two experiments is created by establishing system configuration (defining system configuration designates the allowable types and numbers of components in a system described in Section 6.4.1). Templates for each of the primary component listed are then created.

- CPU
- Graphics Card (GPU)
- RAM
- Motherboard
- Power supply

- Generic software requirements
- Case fan

A balance of systems (BoS) component is also included for representing other parts of the computer for which data was not collected (optical drives, hard drives, ethernet cards, sound cards, etc.). The BoS component provides a lump energy consumption and heat generation approximately equal that of neglected components.

The component models also include interfaces with the system environment including:

- External temperature
- Electrical power rate
- Conversion rate between computations and currency

The component templates are then used for instantiating specific component models (e.g. an NVIDIA GeForce 10 GPU). Component attributes, resource requirements, interface types, and costs for each model are pulled from online databases discussed in Yadav et al. [124] covering the years 2010-2013. In some cases, specific attributes are estimated or interpolated based on the technology Pareto frontier for the specific component. The interface connect and disconnect costs are based on the authors' best estimate of the time and resources required by a professional computer technician.

In total, data was collected for over 60 components. A sample of data collected for RAM is shown in Table 6.1. Similar tables were generated for each component in the system, with approximately 10 models for each component.

Table 6.1: Example RAM Component Information

name	ram: motherboard	throughput (GB/S)	storage_size (GB)	power_use (W)	cost ($)
DDR2-1066-3Gb (2013)	DDR2	8500	3	5	35
DDR2-800-1Gb (2010)	DDR2	6400	1	5	29
DDR3-1066-1Gb (2013)	DDR3	8500	1	4	5
DDR3-1066-3Gb (2013)	DDR3	8500	3	4	15
DDR3-1333-1Gb (2013)	DDR3	10600	1	4	7
DDR3-1333-3Gb (2010)	DDR3	10600	3	4	89
DDR3-1333-3Gb (2013)	DDR3	10600	3	4	21
DDR3-1600-1Gb (2013)	DDR3	12800	1	4	8.3
DDR3-1600-3Gb (2010)	DDR3	12800	3	4	127
DDR3-1600-3Gb (2013)	DDR3	12800	3	4	25
DDR3-1866-1Gb (2013)	DDR3	14900	1	4	9
DDR3-1866-3Gb (2013)	DDR3	14900	3	4	27

Initial system configurations were drawn from the gaming magazine *PC Gamer* [125]. We selected this magazine because it has a recurring build-guide column that includes three tiers of performance (Entry Level, Mid-Range, and Dream) at increasing cost. The components for each build are described in the guide and listed at their then-current market price. We use the guide from the December 2010 issue for creating the three baseline systems, listed in Table 6.2.

For the first experiment, the Entry Level system is used as the baseline with different sized power supplies used for providing tiers of component *excess*. Once a system model is created using specific component models, the cost of the initial system is calculated and passed to the simulation ledger. This is the starting point for the scenario-based simulations.

Table 6.2: System components for the three initial system configurations. Only the Entry Level is used in the first experiment.

	Entry Level	Mid Level	Dream
Power Supply	450W 87% ($95)	750W 87% ($110)	850W 87% ($140)
RAM	DDR2-800-1Gb ($29)	DDR3-1333-3Gb (2x) ($89)	DDR3-1600-3Gb (2x) ($178)
Graphics Card	GeForce GT 220 ($65)	Radeon HD 5870 ($399)	Radeon HD 5870 (2x) ($798)
CPU	Athlon-X2 ($57)	Core I7-920 ($256)	Core I7-950 ($550)
Motherboard	M3A76-CM ($68)	P6T ($244)	P6T Deluxe V2 ($262)
Case Fan	30cfm ($14)	45cfm ($18)	75cfm ($25)
Total Cost	**$328**	**$1116**	**$1953**

6.7.2 Value Assessment and Change Propagation

The value assessment is a simplified version of cryptocurrency mining. An exchange rate is established between the number of Floating-Point Operations the computer performs and dollars. The number of FLOPS required to earn a unit of currency is increased using Equation 6.1 with an initial value of $2.5*10^{-8}$$/GFLOP.

$$R_t = R_0 * e^{\frac{-t}{t_d}*\ln(2)} \tag{6.1}$$

where R_t is the conversion rate at time t, R_0 is the initial conversion rate, t is number of elapsed time periods, and td is the halving time. The recurring cost for the system is the cost of electric energy consumed. The monthly energy use is calculated from the system model and expensed at a rate of $0.11/kw*hr, which increases by a fixed percentage each year as noted in the description of each experiment. Both costs and value generated are discretized monthly.

An epoch shift occurs once the simulation reaches 2014. The system can either be left as-is or upgraded. If left as-is, the same system is simulated for another three years. Otherwise a change process is initiated. The change process is initiated with the total computation capacity node as the change initiator.

Candidate change pathways are generated as described in Section 6.6. A list of all possible combinations (between 15,000 and 50,000 depending on the initial system's motherboard model) of available components of each type is assembled and a new system is generated with each. If the new system has no violated requirements, it is simulated for a second epoch.

If requirement violations occur (e.g. insufficient power supply capacity), the violated requirement becomes the target for another round of change pathway generation. This process

repeats so long as unexercised options remain. If no suitable configuration can be found, the system is discarded.

The total change cost for the upgraded system is determined by the disassembly, new component purchases, and reassembly costs for the upgraded system. This sum is added to the ledger on the epoch shift date. At the end of the second epoch the cryptocurrency value and power costs for each system are discounted to 2010 and tallied.

6.7.3 Experiment One – Value of Excess in a Component

The first experiment focuses on the value of *excess* in a single component. The hypothesis for this experiment is that *some* excess power supply capacity increases the value of the system. The power supply does not directly contribute to system computation, but does supply more power than required in the initial system. Including a larger power supply could potentially reduce the cost of future changes. We test this hypothesis using the Entry Level computer with five different initial systems, each with a different capacity power supply. The lifecycle values of each are assessed and compared. *This experiment demonstrates that the methodology can identify optimal component sizing for individual components.*

6.7.3.1 Setup

The base system for this experiment is the Entry Level system from Table 6.2. Five levels of power supply capacity are used: 450W, 750W, 850W, 1000W, and 1350W. The halving time (t_d) for the conversion between FLOPS and dollars was set to 24 months, the annual power cost increase was set to 3%, and the discount rate was set to 5% per year. For each power supply capacity, lifecycle values and component commonality scores are averaged across all final configurations.

6.7.3.2 Results

Simulation results of averaged system lifecycle value and average number of common components are shown in Figure 6.11. As the power supply increases in size, the average number of components in common increases. This occurs because the larger supply can support more system configurations. The commonality increases as capacity increases until the power supply exceeds 1000W. The 1000W threshold is sufficient for satisfying the requirements for most feasible rebuilds. This is important, as the power supply would not have to be replaced when the system is modified.

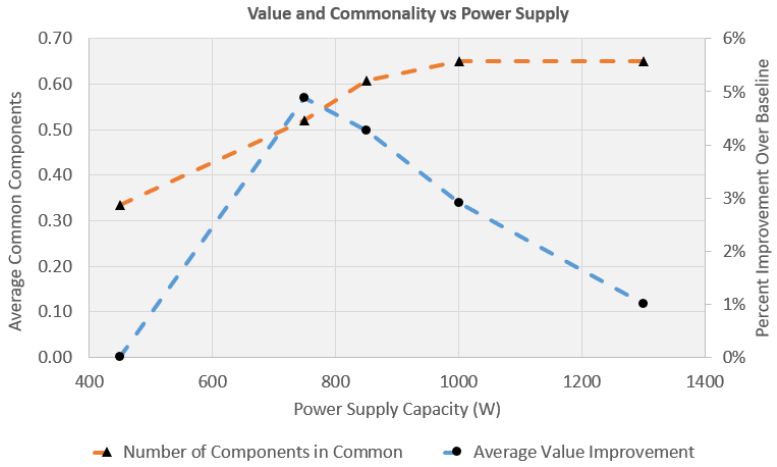


Figure 6.11: Larger initial power supplies increase component commonality but that lifecycle value improvement peaks at 750W

When considering lifecycle value, power supplies that are too large or too small are suboptimal. Too small and the supply must be replaced for new system configurations. Too large and there is risk that the extra cost incurred does not yield enough benefit in subsequent configurations. The performance of each power supply in plotted in Figure 6.11. The trendlines are linear approximations between data points. Given the capacities studied, the optimal power supply size is 750W.

6.7.4 Experiment Two - Value of Excess in a System

All three configurations in Table 6.2 are now used. *The tested hypothesis is that the Mid Range system provides the best lifetime value.* This hypothesis is based on the assumption that more powerful systems provide greater potential for value generation (while potentially consuming more electric energy) and theoretically require fewer changes during the change step. However, more powerful systems also have significantly higher initial costs. *This test provides insight into the balance between initial system investment, the return on that investment, and the variables that influence that balance.*

6.7.4.1 Setup

We also test the sensitivity of the simulation to exogenous variables by varying three parameters: the halving time for the cryptocurrency exchange rate, the percent annual electricity rate increase, and the discount rate. The levels used in testing are reported in Table 6.3.

Table 6.3: Exogenous variable parameters and their levels

	Low	Medium	High
Halving Time (months)	30	24	18
Annual Electricity Rate Increase (%)	1	3	5
Discount Rate (%)	5	n/a	12

6.7.4.2 Results

We generated ~870,000 valid combinations of system configurations and exogenous variable scenarios. A violin chart showing the distribution of lifecycle values for each initial computer type, halving time, and discount rate is shown in Figure 6.12. The annual electricity rate increase levels are not reported because they marginally impact the results. Mean lifecycle values for each computer - exogenous variable combination are reported in Table 6.4.

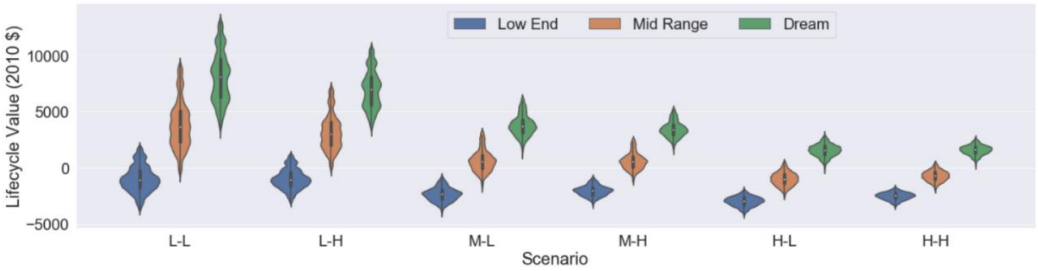


Figure 6.12: System lifecycle value distribution by computer type and scenario (Halving Time, Discount Rate) showing the sensitivity of the outcomes to each category. The halving time and the initial computer appear to have the greatest influence.

The largest impact on lifecycle value come from the initial computer configuration and cryptocurrency halving time. The mean lifecycle value of the Dream computer is $6212 higher than the Entry Level computer. This means that the extra initial investment for higher performance is the optimal choice. However, in Figure 6.12, we see that the Dream computer is more sensitive to exogenous variables and rebuild component selection (a standard deviation of $2801 compared against $1034 for the Entry level computer).

Lifecycle values were most sensitive to the halving time parameter governing the cryptocurrency exchange rate. The difference in mean lifecycle value between low (30 months) and high (18 months) levels of the halving time parameter is $2723. Figure 6.12 also shows that the dispersion of lifecycle values for the high level is smaller than those of the lower level As halving time affects the marginal value of added computational power, longer halving times significantly widen the value gap between Dream and Entry systems.

Table 6.4: Mean lifecycle value of all simulated systems for simulations using tested parameters showing higher sensitivity to Computer Type and Halving **Time**

Mean Lifecycle Value vs Parameter Levels (2010 $)

Parameter Level	Low	Medium	High
Computer Type	-1970	1094	4242
Halving Time	839	-1003	-1884
Annual Electricity Rate Increase	-500	-679	-869
Discount Rate	-731	n/a	-635

Lifecycle values were less sensitive to discount rate and annual electricity rate increase. The primary effect of the higher discount rate is decreasing the relative contribution of the rebuild and second epoch. The costs and generated value later in the simulation are smaller with a higher discount rate. The annual electricity rate increase decreases the lifecycle values for all systems with larger decreases for more energy intensive systems.

A consequence of higher electrical energy cost rates is that more energy efficient systems have relatively better lifecycle values. For example, in the highest power cost scenarios the GPUs of 8 of the 15 highest lifecycle value systems use the most power efficient GPU, versus 4 of 15 systems in scenarios with the lowest power cost.

These results do not support the hypothesis stated for this experiment. To understand why, we conducted an analysis of which systems were optimal in the rebuild. Assessing the simulation from rebuild to system retirement requires a new measure called the Epoch 2 (E2) Value. The formulation of this metric is shown in Equation 6.2.

$$E2Value = E2CryptocurrencyValue - RebuildCost - E2ElectricityCost \quad (6.2)$$

This measure excludes the initial system cost along with all operating costs and value generated during the first epoch. This allows a more forthright assessment of which component choices are optimal for the rebuild and how well *excess* supports those choices.

Using this metric, we tested the assumption that *excess* in the initial system would result in fewer replaced components during the rebuild. The set of systems included for this analysis is reduced by only including the top performing system for each combination of computer type, scenario, and number of common components. Figure 6.13 shows the distribution of E2 Value grouped by the number of components in common between the initial and rebuild systems. Fewer components in common result in superior E2 Value, which is counter to the assumption.

Systems with zero or one components in common have the best E2 Value, while systems with a majority of components in common have overall worse performance.

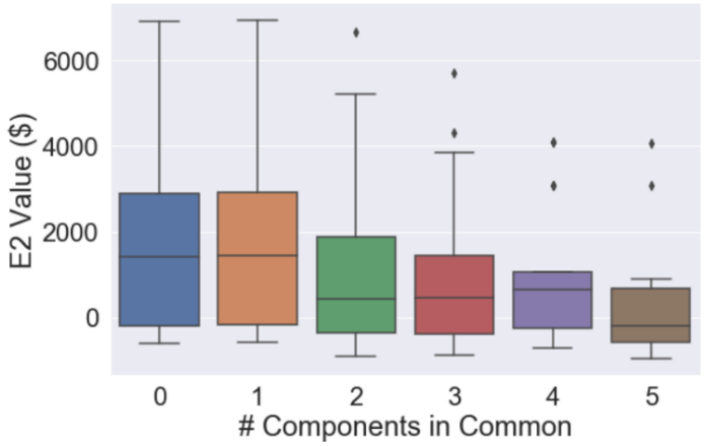


Figure 6.13: A comparison of E2 Total Cost and number of common components showing decreasing value as commonality increases.

Understanding why this result happens requires analyzing which component types and component models are found in high performance systems. This is done in Figure 6.14 by averaging the E2 Value for all systems containing a specific component model (e.g. all computers containing an i7-950). Each mean is then represented by a single dot in each column of Figure 6.14.

An additional column representing the number of GPUs is also included as the number of GPUs has a significant impact on E2 Value. For example, the four dots shown for GPU# are (from bottom up): 4, 1, 2, and 3 GPUs included in the system. The goal of this chart is depicting the sensitivity of E2 Value to changes in components. The horizontal offset within a column is used for visual distinction of similar means.

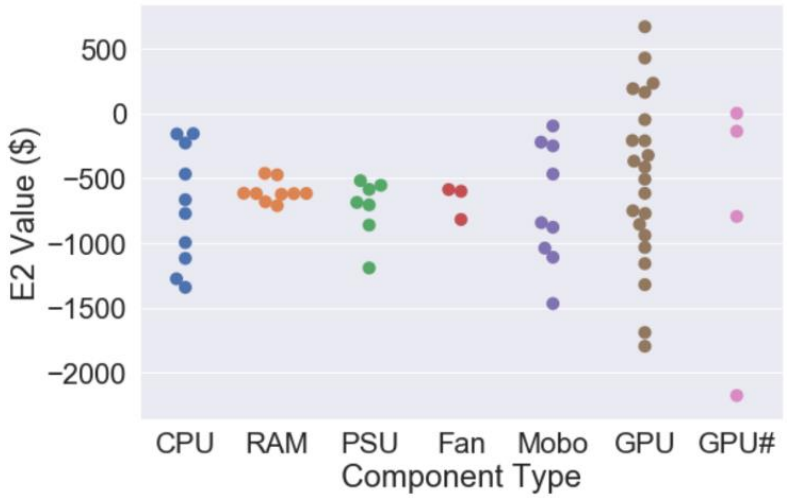


Figure 6.14: E2 costs vs component instance showing that the type and number of GPU used has the largest impact on system value.

The larger vertical dispersion of components in the GPU and GPU# columns mean that these attributes have the greatest degree of influence on E2 Value. The GPU is generally the most expensive component to replace and has the largest influence on the cryptocurrency generation rate. The GPU therefore contributes more substantially to the E2 Value than other components. The motherboard and CPU have the next most significant influence. The CPU degree of influence is larger because it contributes directly to computational power, which is one constraint on the maximum computation of the GPU(s).

The motherboard is significant because it limits both the allowable CPU models via the CPU slot type and sets the maximum number of GPUs via the number of GPU slots available. Interestingly one of the most predictive indicators of system value, given the motherboard, is the CPU slot type. The motherboards that support the Intel LGA1150 slot provide access to CPUs with the largest computational power at the time of the rebuild. The motherboards with the LGA1150 slot therefore outperform other motherboards despite the premium in motherboard cost because the associated CPUs have superior computational capacity.

Ultimately the reason that computers with little commonality have better E2 Values is due to a three-link casual chain:

1. The systems with the best E2 Total Cost values are those that contain multiple graphics cards with the largest FLOPS attribute.
2. These systems require more CPU computational capacity (more than included for any initial system) to prevent CPU locking (where the CPU is unable to fully task GPUs).
3. The motherboard socket types used in the initial system do not support CPUs released in the rebuild year. Since new CPU's are required to support the more demanding GPU load the motherboards must be replaced.

This suggests that three years may be too long between rebuilds for this scenario. More frequent rebuilds would allow an incremental approach to upgrading and potentially avoid the scenario where the best course of action is replacing the entire system. The combined problem of optimal upgrade strategy and component replacements will be addressed in future work.

While the strategy of waiting three years for an upgrade did not support the inclusion of significant excess, the results do suggest that, under different scenarios, excess might be a valuable tool for reducing upgrade costs. Excess may be used for interrupting the causal chain described above by either shortening the upgrade window or selecting a motherboard with a socket earlier in its development window. In retrospect, the LG1366 socket in both the Mid and Dream computers was near end of life with retirement at the beginning of 2012. With the improved understanding of how change propagates from GPU to motherboard, it might be prudent to find a socket capable of supporting new CPUs longer. This would prevent the need for motherboard replacement, saving hundreds of dollars.

This leads to the most important conclusion. This model captures underlying system interactions suggesting both where excess might be included (the motherboard) and **how** to incorporate it (have a longer-lived socket).

6.8 Chapter Summary

This chapter demonstrates how the Decision Based Design (DBD) approach can be used for assessing a system's lifecycle value when including *excess*. While the example presented is relatively simple and the technology trajectories known with certainty, the combination of *excess* and change propagation within a DBD based framework is a promising path for rigorously assessing the value of changeability enabled by *excess*. This has not been possible with any prior research.

The most salient results of this paper are that the approach used can provide significantly more nuanced results than existing graph-based change propagation models with a moderate increase in model complexity. The current approach not only predicts which change propagation pathways are likely to be experienced but also suggests what components might prevent change propagation (demonstrated in experiment 2) AND what degree of component sizing is optimal (demonstrated in experiment 1). Combined these offer designers a much clearer picture of how *excess* affects system lifecycle value than current change propagation methods.

This work advances the state-of-the-art by the introduction of a novel system representation framework. The new representation improves existing flexibility and excess research by:

- reducing the degree of subjectivity required for assessing the value of changeability and excess,

- automating the link between system representation and lifecycle model and thereby improving system optimization (experiment 1) and,
- improving flexibility assessment by identifying both where (in which components) flexibility is beneficial and how flexibility can be included in in that component to improve lifecycle value (exper

  iment 2).

This work provides a foundation for evaluating how and where *excess* should be included by combining system lifecycle evaluation with change propagation. Doing so allows the simulation of potential changes and lifecycle trajectories. This foundation is enabled by the development of a system representation that can automatically execute both primary changes and subsequent change propagation and estimate system lifecycle performance including those changes.

Chapter 7: Conclusions and Future Work

7.1 Research Summary

The introduction of this dissertation stated that systems are more complex, more interconnected, and use components which have rapidly changing technology. Engineers must design systems within this context while maximizing system lifecycle value given the uncertainties associated with future needs and requirements. Changeability reduces the effort associated with modifying the system in the future. While changeability may improve system lifecycle value when used well, when not used appropriately it may reduce initial system performance and increase costs with little future benefit. This research began by studying how excess has been used in aircraft, desktop computers, and gaming consoles (Chapters 3 and 4). These case studies showed that excess can improve system changeability and that excess can improve overall system lifecycle value. New ideas and methods for assessing the impact and value of excess are then introduced by extending tools found in existing literature (Chapter 5) and by building on an existing framework using modern computational tools (Chapter 6). Incorporating changeability and excess requires significantly more research, but the ideas and approaches in this research advance the boundaries of existing literature.

This remainder of this section has two parts. The first reviews and discusses the initial research questions in the context of the results and findings from Chapters 3-6. The second part addresses the need for further system changeability research by discussing potential directions for future research efforts.

7.2 Discussion of Research Questions

Research Question 1) What system design lessons can be learned from qualitative evidence linking the presence/absence of excess and system lifecycle value?

The results of Chapter 3 suggest that excess does influence system lifecycles for the studied systems. The presence, or lack, of excess was found to have profound consequences for the lifecycle of the two aircraft studied.

The B-52 possesses ample excess in size, internal volume, and range attributes enabling the USAF to adapt the aircraft to emergent needs for decades. These include adaptations for low-level flight and electronic countermeasures to counter surface to air missiles, expanded internal and external carrying capacity to support conventional ordinance, and integration of modern electronics to carry guided ordinance and integrate it with modern command and control infrastructure.

In contrast, the F/A-18 Hornet lacked sufficient excess for supporting desired modifications. Government records were uncovered discussing design decisions and highlighting exactly which shortcomings necessitated an expensive redesign. These shortcomings were associated with an absence of excess in critical aircraft components. The Hornet was redesigned as the Super Hornet - addressing the absence of excess by adding fuel capacity, internal volume, and wing area addressing the shortcomings of the original design.

Together these two case studies support the hypothesis that presence (or absence) of excess influences system lifecycle performance. Four design lessons were observed from the results of the qualitative study.

1) Modularity alone may be insufficient for making a system changeable. Modifications often also require supporting excess be available.
2) System requirements are stochastic in nature. The presence of excess improves system changeability for better meeting those requirements.

3) System excess exhibits a degree of fungibility where excess in one area may be traded for excess in another.
4) The magnitude of excess required for supporting modifications should be considered with respect to the magnitude of excess present within the system.

A system possessing excess may be adapted more easily to new requirements than one that does not. Chapter 3 did not, however, discuss the overall optimality of including excess. The argument could be made that the decades of service for the B-52 do not sufficiently recover the large initial investments in the design and production of the aircraft or that the lower initial cost of the Hornet, and the cost of the redesign, were less costly than initially designing in more excess. Chapters 4 and 6 address system lifecycles in a more holistic way, but this is an area where future research is required as described in Section 7.3.

Research Question 2) What system design lessons can be learned from quantitative evidence linking the presence/absence of excess and system lifecycle value?

Several lessons with respect to excess and system design are introduced in Chapter 4. First excess can improve future requirement satisfaction, but an increase in system utility depends on the benefit derived by the user depending on their preferences. Excess enhances system value under some preference sets, while in others it does not. Excess value assessment is therefore nuanced, requiring knowledge of how excess could be included in the initial system, a forecast of future uncertainties, and knowledge of user preferences.

Excess assessment at the system level showed that computers with more excess in all components had longer useful lifetimes and were able to play more games with more demanding settings. This aligns with findings from Chapter 3, but an additional assessment was conducted incorporating the initial costs of the systems. Different degrees of initial system excess and cost

were shown to be optimal under different user preference assumptions. Customers who prefer to play the most demanding games derived the most value from the computer with the best initial performance and highest initial cost. Customers who prefer only to play the largest number of games with a stronger preference for lower costs were best served by the lowest performance and least expensive computers.

Additionally, the study is Chapter 4 suggested that allowing customization of the type and degree of excess by the consumer improves system performance. A comparison between consoles (for which excess decisions are made by the manufacturer with infrequent generation updates) and desktops (with frequent generation updates) found that console performance was only competitive for the first few years of a new generation, after which better performance was provided by purchasing a desktop with the most recent technology included. Further customization by improving systems via strategic excess in components critical for future success also improves a system's ability to satisfy future requirements.

Strategic excess was shown to improve system performance. For example, improving system RAM (average cost 7% of system) improved performance (number of playable games) by an average of 14%. This suggests that adding excess to components strategically may offer performance improvements without the cost of adding excess to all components and that finding these key components may be central for cost effective use of excess for improving system lifecycle performance.

Research Question 3) What extensions are needed in existing change propagation methods so that excess can be incorporated, and its effect modeled?

Chapter 5 described an extension to existing change propagation tools that assumed a relationship between component excess and change propagation tendency. The resulting model

provides general guidance for excess placement within a system based on the system architecture (connections between components) useful for high-level design at the concept development stage and we demonstrated on several systems found in existing literature.

Certain architecture patterns were found that supported system changeability (those incorporating system hubs – especially out-degree hubs). Analysis of a system from literature with the extended method provided further evidence that key components have an outsized influence on a system's changeability. These key components tend to have higher numbers of connections with other components and connect to other highly connected components. Identifying potential key components based on the system architecture alone is a good first step towards improving system changeability, but without more information about how components contribute to system performance and requirement satisfaction it remains a largely qualitative solution.

Integrating excess into existing tools provides a useful method with practical but limited application. However, the inability to incorporate change costs or differentiate between different kinds of component excess emphasized the need for an entirely new approach that explicitly includes excess capable of supporting a detailed system lifecycle simulation.

Research Question 4) What phenomenon must be included for modeling the effects of excess and how can they be combined for creating a holistic assessment of excess location and degree on system lifecycle value?

The challenge in forecasting lifecycle value when incorporating excess is accounting for the interdependencies between components, system-level attributes, operational costs, and generated value.

The approach taken in Chapter 6 uses using an existing framework, Decision-Based Design, for modeling all aspects of the system pertinent to its lifecycle value. A flexible system representation is implemented using SysML-inspired component blocks with embedded computational networks. The result is an approach and method capable of providing significantly more nuanced results than existing graph-based change propagation models by including the types of excess, requirements associated with excess that trigger required changes, and links between component attributes with aggregate system-level attributes. This system model is then used in tandem with simulated lifecycles for comparing the value of different initial system designs. This is accomplished with a moderate increase in model complexity.

The approach was demonstrated by assessing the lifecycle value of desktop computers tasked with cryptocurrency mining. The first experiment showed that the optimal power supply for the computer balanced initial cost with providing sufficient power supply capacity to support future modifications. The second experiment compared the lifecycle value of three distinct systems for a period of six years with a rebuild in year 3. It was found that the key component attribute for system changeability in this example was the CPU socket type for the motherboard. The experiments demonstrate the new approach can identify what components might prevent change propagation (experiment 2) AND what degree of component sizing is optimal (demonstrated in experiment 1). This represents a significant advance in assessing the value of excess and system changeability generally by accounting for the myriad interconnections which influence the value of specific kinds of excess within a system.

7.3 Future Research Questions

The prior research presented in Chapter 1 indicates system changeability has and will continue to play a central role in enhancing system lifecycle value when compensating for

uncertain future requirements. Chapters 3 and 4 provided evidence linking excess with system performance and value that builds on existing work by others described in Section 2.7. Chapters 5 and 6 presented approaches for modeling excess by extending existing changeability research and by using an existing design approach with a modern implementation. Understanding and modeling excess and changeability is still incomplete, and requires significantly more research to make the transition suggested by Saleh et al. [4] from "vague and difficult to improve, yet critical to competitiveness" to an established ontology with a core of accepted approaches. The remainder of this section describes avenues of future research that strengthens evidence for how excess influences system performance, improves excess and changeability modeling, and uncovers how practicing engineers think about, and work with, excess.

7.3.1 Exploring Excess in Practice

As suggested by Eckert et al. [16] excess and margins are currently hidden issues in industry. Evidence presented in Chapter 3, and in the Eckert el al. study, indicates that companies are likely using excess without formally modeling or assessing it. Assigning quantities of excess in the design process is likely based on expert assessment from experienced engineers. One goal of future research is assessing how experts decide how much excess should be incorporated. There is more to be learned by observing how excess has been, or is currently being, used and how decisions about excess are made. A study should be conducted assessing if - and how - engineers use excess, if they recognize excess within a system, and if management uses knowledge of excess in decision making. This should include a study of the relationships between uncertainty and excess since improved modeling may be used to reduce uncertainty and convert buffer into excess.

Data collection would require a company willing to allow access to its design process and its employees. Optimally a system that has been modified over time would be used as a test case. The original system would first be assessed for the presence of excess (using an approach similar to Chapter 6). A post-mortem of historical modifications would then be conducted with both the engineering and management team building an understanding of how each group understands and interacts with system excess.

An additional study could be conducted with engineering students. A portion of the students would be introduced to excess and changeability with the remainder used as a control group. A simplified system and value proposition would be presented, and each student group asked to design a system with the goal of maximizing system lifecycle value. The results between the two groups would then be compared to assess whether the concepts improved outcomes.

7.3.2 Value and Risk Modeling

Two of the most limiting assumptions from Chapter 6 are the dependencies on technology forecasting and value modeling. The study in Chapter 6 simplified these by using historical technology trajectories and a simplified time-dependent conversion between system performance and system value. Chapter 5 addressed the value issue by demonstrating how different customer preference models would affect system value. Understanding the uncertainty from each and how they create decision making risk is needed future work.

Modeling system value for consumer products requires introducing a demand model that accounts for customer preferences, competitor products, and the firm's portfolio of other systems. Similar work has been done with product platforming (as introduced in Section 2.4) which may be used as a guide for incorporating excess into a firm's design process. System's

without clear value models (e.g. military systems and systems that conduct scientific research) may require a utility-based measurement. In each case excess models must be linked into existing processes requiring additional research.

Another external change driver influencing system value is the interconnectedness between the system of interest and interfacing systems (those outside the control volume of the system). This was discussed in Chapter 4 regarding the influence of the display used by the desktop and console. Desktops traditionally use computer monitors that historically have higher resolutions and faster refresh rates than the televisions used for consoles. This distinction was highlighted as a possible reason console GPU performance was acceptable despite its difficulty satisfying PC game requirements near end-of-generation – the display requirements for consoles were less demanding. Other interfaces may also have significant consequences for system design. For example, the introduction of 3D headsets very likely to require far more GPU computation than current 2D display technologies. This is likely to drive future GPU performance requirements, and possibly the design of the entire computer, to a much greater extent than incremental improvements in existing game requirements. Beyond computers/consoles there is the need for expanding excess-focused research into the design of Systems-of-Systems [126], an environment where multiple, independently operating systems work together toward a common goal. Understanding the value of excess in this environment has not been studied yet offers substantial value opportunities especially when system upgrades are considered.

While improvements in value modeling and technology forecasting can reduce epistemic uncertainty there will always be some uncertainty when forecasting future states of technology or value. Adding excess in a system adds cost, but the future payoff is uncertain. Creating

simulations with different future states (as was done in Chapter 6) can assess how well each initial system performs in each, but not all future states are equally likely. A key future research area is therefore determination and communication of risk associated with added excess and the uncertainty associated with specific forecasted exogenous variables.

7.3.3 Software-Centric Systems

Modern electromechanical systems include both hardware and software. The cost of modifying software is far less than making physical changes to the system. This introduces a low-cost way of updating a system after it has been placed in service. This practice is common for computers and smart phones but is increasingly used for other systems. The auto manufacturer Tesla and its efforts to develop autonomous driving capability provides an ideal test case for further examination of how hardware excess can be used by future software updates.

Tesla is facing the challenge that the hardware required to support the transition to a fully autonomous system is advancing at a pace faster than the car's lifetime and the software is advancing even more quickly. Designing a system that relies on components and algorithms that quickly become obsolescent is a major challenge. Designers must allow for a reasonable degree of software and hardware growth supporting the increasing requirements of the autopilot system. Tesla's strategy appears to be including excess in sensor hardware and computing power.

In 2016, Tesla equipped all new cars with the Enhanced Autopilot Hardware (EAH) 2.0 suite of sensors and processors including 7 new cameras, sonar sensors with extended range, and a new (replaceable) computing platform. At deployment, the software was only capable of supporting driver-assist capabilities similar to the original 2014 Autopilot. The company marketed EAH 2.0 as one day being capable of fully autonomous driving. This implies that enough excess was included to support all future software upgrades. The degree to which the

sensors were included as excess was revealed when testing showed the software was making use of only 1 of the 8 total cameras as input [127]. Following a 2016 hardware rollout, the software has gradually grown more capable and is transitioning towards full autonomy requiring the data from the once unused cameras.

The study of Tesla's autopilot system demonstrates that excess was used to minimize the modifications required when updating fielded systems. Tesla incorporated more sensors than were required for the initial autopilot system and gradually consumed the excess as the autopilot algorithms improved. Tesla's engineers also attempted to include sufficient excess in the onboard computing capacity to accommodate the transition to autonomous driving but hedged the risk by making the onboard computer replaceable. In 2019 Tesla engineers realized that fully autonomous driving would require additional computational power and offered to replace the onboard computer for some older models. The prior decision to ensure the processor was replaceable likely reduced the ensuing replacement cost.

A detailed case study of the Tesla autopilot hardware and software deployment strategy could prove highly illustrative in understanding and modeling how hardware excess was used to support future software updates which improve the value of fielded systems at marginal cost.

7.3.4 The Coupled System/Strategy Problem

One difficulty inherent in selecting an optimal design is that the system initial design is coupled with the system operator's lifecycle decision making and relevant exogenous parameters. To faithfully model the impact excess inclusion can have on a system a lifecycle, the decisions made once the system is operational need to be appropriate for changing circumstances. Retiring a system early with unused excess will not fully take advantage of included excess. The same is true if a system is pushed beyond its reasonable capacity for

upgrade. The future work is therefore focused on how exogenous variables change and what actions the system operator should take.

Part of what is necessary is to encode a set of rules that govern the transitions for uncertain variables. This could include GPU/CPU performance, crypto-currency value, or the cost of electric power. The transitions could be a random walk, binomial lattice [119] or some other mechanism as appropriate. The second necessary addition is the set of actions that the designers are allowed to take. For the example problem this could include:

- Upgrading one or multiple components in the system
- Retiring the system
- Exercising an available option (like purchasing an additional GPU for an empty slot)
- Nothing

As shown in Chapter 6, each action would have some effect that is governed by the change dynamics of the system. The goal is then to develop a policy that generates optimal decision making given the above action and transition dynamics. A number of algorithmic alternatives exist for this portion. A small enough problem with a sufficiently coarse time discritization could be solve via exhaustive enumearation (as done in Chapter 6) or dynamic programming. Sufficiently complex problems could require reinfocement learning techniques and possibly involve approximate solution methods as outlined in Sutton and Barto [128].

REFERENCES

[1] Simpson, T. W., and Martins, J. R. R. A., 2011, "Multidisciplinary Design Optimization for Complex Engineered Systems: Report From a National Science Foundation Workshop," J. Mech. Des., **133**(10), p. 101002.

[2] Bloebaum, C. L., and McGowan, A.-M. R., 2012, "The Design of Large-Scale Complex Engineered Systems: Present Challenges and Future Promise," AIAA Aviat. Technol. Integr. Oper. Conf. 14th AIAA/ISSM, (September), pp. 1–19.

[3] Collopy, P. D., and Hollingsworth, P. M., 2012, "Value-Driven Design," J. Aircr., **48**(3).

[4] Saleh, J. H., Mark, G., and Jordan, N. C., 2009, "Flexibility: A Multi-Disciplinary Literature Review and a Research Agenda for Designing Flexible Engineering Systems," J. Eng. Des., **20**(3), pp. 307–323.

[5] United States Governmenent Accountability Office, 2015, *Assessments of Selected Weapon Programs.*

[6] Capaccio, T., 2019, "F-35 Fighters Will Cost $22 Billion More Than Expected, Pentagon Says," Time Mag. [Online]. Available: https://time.com/5575608/lockheed-martin-f-35-jet-cost/. [Accessed: 09-Jun-2019].

[7] United States Government Accountability Office, 2016, "James Webb Space Telescope: Project Meeting Cost and Schedule Commitments but Continues to Use Reserves to Address Challenges," (December), p. 33.

[8] Jeff Foust, 2019, "JWST and SLS Drive up Cost and Schedule Growth on NASA Programs - SpaceNews.Com," Sp. News [Online]. Available: https://spacenews.com/jwst-and-sls-drive-up-cost-and-schedule-growth-on-nasa-programs/. [Accessed: 06-Sep-2019].

[9] Loren Grush, 2018, "Why NASA Is Struggling to Get Its Most Powerful Space Telescope

off the Ground - The Verge," The Verge [Online]. Available: https://www.theverge.com/2018/8/1/17627560/james-webb-space-telescope-cost-estimate-nasa-northrop-grumman. [Accessed: 06-Sep-2019].

[10] Tumer, I. Y., and Lewis, K., 2014, "Design of Complex Engineered Systems," Artif. Intell. Eng. Des. Anal. Manuf. AIEDAM, **28**(4), pp. 307–309.

[11] Voosen, P., 2009, "How Long Can a Nuclear Reactor Last? - Scientific American," Sci. Am. [Online]. Available: https://www.scientificamerican.com/article/nuclear-power-plant-aging-reactor-replacement-/. [Accessed: 03-Jan-2017].

[12] Swarts, P., 2016, "Air Force Prolongs the Life of the Venerable B-52," Air Force Times.

[13] Heisler, T., 2014, *C-130 Hercules : Background , Sustainment , Modernization , Issues for Congress*.

[14] Fricke, E., and Schulz, A. P., 2005, "Design for Changeability (DfC): Principles to Enable Changes in Systems throughout Their Entire Lifecycle," Syst. Eng., **8**(4).

[15] Schulz, A. P., Fricke, E., and Igenbergs, E., 2000, "Enabling Changes in Systems throughout the Entire Life-Cycle – Key to Success ?," Proc. 10th Annu. INCOSE Conf., (July), pp. 565–573.

[16] Eckert, C., Isaksson, O., and Earl, C., 2019, "Design Margins: A Hidden Issue in Industry," Des. Sci., **5**(May), p. e9.

[17] Allen, J., Mattson, C. A., Magleby, S. P., Howell, L. L., Mclain, T. W., and Fullwood, D. T., 2016, "Evolvability and Excess Capability as a Response to Uncertain and Future Requirements."

[18] Maier, M., and Rechtin, E., 2000, *The Art of Systems Architecting*, CRC Press, Boca Raton.

[19] Ulrich, K., 1995, “The Role of Product Architecture in the Manufacturing Firm,” Res. Policy, **24**(3), pp. 419–440.

[20] Eckert, C., Clarkson, P. J., and Zanker, W., 2004, “Change and Customisation in Complex Engineering Domains,” Res. Eng. Des., **15**(1), pp. 1–21.

[21] Keese, D. A., Seepersad, C. C., and Wood, K. L., 2006, “An Enhanced Change Modes and Effects Analysis (CMEA) Tool for Measuring Product Flexibility With Applications to Consumer Products,” *IDETC/CIE*, p. 16.

[22] Pasqual, M. C., and De Weck, O. L., 2012, “Multilayer Network Model for Analysis and Management of Change Propagation,” Res. Eng. Des., **23**(4), pp. 305–328.

[23] Clarkson, P. J., Simons, C., and Eckert, C., 2004, “Predicting Change Propagation in Complex Design,” ASME Des. Eng. Tech. Conf., **136**(August 2004), pp. 788–797.

[24] Martin, M. V, and Ishii, K., 2002, “Design for Variety: Developing Standardized and Modularized Product Platform Architectures,” Res. Eng. Des., **13**(3), pp. 213–235.

[25] Engel, A., and Browning, T. R., 2016, “Designing Products for Adaptability : Insights from Four Industrial Cases Designing Products for Adaptability : Insights from Four Industrial Cases.”

[26] Robertson, D., and Ulrich, K., 1998, “Planning for Product Platforms,” Sloan Manag. Rev., **39**(4), pp. 19–31.

[27] de Weck, O. L., de Neufville, R., and Chaize, M., 2003, “Enhancing the Economics of Communications Satellites via Orbital Reconfigurations and Staged Deployment,” AIAA Pap., **6317**(September), p. 2003.

[28] Hazelrigg, G. a., 1998, “A Framework for Decision-Based Engineering Design,” J. Mech. Des., **120**(4), p. 653.

[29] Hamraz, B., Caldwell, N. H. M., and Clarkson, P. J., 2013, "A Holistic Categorization Framework for Literature on Engineering Change Management," Syst. Eng., **16**(4), pp. 473–505.

[30] Cardin, M.-A., 2013, "Enabling Flexibility in Engineering Systems: A Taxonomy of Procedures and a Design Framework," J. Mech. Des., **136**(1), p. 011005.

[31] Ferguson, S., Siddiqi, A., Lewis, K., and de Weck, O. L., 2007, "Flexible and Reconfigurable Systems: Nomenclature and Review," *Volume 6: 33rd Design Automation Conference, Parts A and B,* ASME, pp. 249–263.

[32] Suh, N. P., 1998, "Engineering Design Axiomatic Design Theory for Systems," Res. Eng. Des., **10**(4), pp. 189–209.

[33] Hölttä-Otto, K., Chiriac, N. A., Lysy, D., and Suk Suh, E., 2012, "Comparative Analysis of Coupling Modularity Metrics," J. Eng. Des., **23**(10–11), pp. 790–806.

[34] Gershenson, J. K., Prasad, G. J., and Zhang, Y., 2003, "Product Modularity: Definitions and Benefits," J. Eng. Des., **14**(3), pp. 295–313.

[35] Jiao, J., Simpson, T. W., and Siddique, Z., 2007, "Product Family Design and Platform-Based Product Development: A State-of-the-Art Review," J. Intell. Manuf., **18**(1), pp. 5–29.

[36] Simpson, T. W., Bobuk, A., Slingerland, L. A., Brennan, S., Logan, D., and Reichard, K., 2012, "From User Requirements to Commonality Specifications: An Integrated Approach to Product Family Design," Res. Eng. Des., **23**(2), pp. 141–153.

[37] Sosa, M. E., Eppinger, S. D., and Rowles, C. M., 2007, "A Network Approach to Define Modularity of Components in Complex Products," J. Mech. Des., **129**(11), p. 1118.

[38] Suh, E. S., De Weck, O. L., and Chang, D., 2007, "Flexible Product Platforms:

Framework and Case Study," Res. Eng. Des., **18**(2), pp. 67–89.

[39] Tilstra, A. H., Backlund, P. B., Seepersad, C. C., and Wood, K. L., 2015, "Principles for Designing Products with Flexibility for Future Evolution," Int. J. Mass Cust., **5**(1), pp. 22–54.

[40] Hazelrigg, G. A., 2012, *Fundamentals of Decision Making for Engineering Design and Systems Engineering*, Pearson Education Inc.

[41] Fixson, S. K., 2004, "Assessing Product Architecture Costing: Product Life Cycles, Allocation Rules, and Cost Models," *Proceedings of the ASME Design Engineering Technical Conference*, pp. 857–868.

[42] Lee, B. D., and Paredis, C. J. J., 2014, "A Conceptual Framework for Value-Driven Design and Systems Engineering," Procedia CIRP, **21**, pp. 10–17.

[43] Simpson, T. W., Maier, J. R., and Mistree, F., 2001, "Product Platform Design: Method and Application," Res. Eng. Des. - Theory, Appl. Concurr. Eng., **13**(1), pp. 2–22.

[44] Simpson, T. W., 2004, "Product Platform Design and Customization: Status and Promise," Artif. Intell. Eng. Des. Anal. Manuf. AIEDAM, **18**(1), pp. 3–20.

[45] Koh, E. C. Y., Caldwell, N. H. M., and Clarkson, P. J., 2013, "A Technique to Assess the Changeability of Complex Engineering Systems," J. Eng. Des., **24**(7), pp. 477–498.

[46] Giffin, M., de Weck, O., Bounova, G., Keller, R., Eckert, C., and Clarkson, P. J., 2009, "Change Propagation Analysis in Complex Technical Systems," ASME J Mech Des., **131**(August), pp. 1–14.

[47] Suh, E. S., De Weck, O. L., and Chang, D., 2007, "Flexible Product Platforms: Framework and Case Study," Res. Eng. Des., **18**(2), pp. 67–89.

[48] Tilstra, A. H., Backlund, P. B., Seepersad, C. C., and Wood, K. L., 2015, "Principles for

Designing Products with Flexibility for Future Evolution," Int. J. Mass Cust., **5**(1), pp. 22–54.

[49] Tackett, M. W. P., Mattson, C. A., and Ferguson, S. M., 2014, "A Model for Quantifying System Evolvability Based on Excess and Capacity," J. Mech. Des., **136**(5), p. 051002.

[50] Allen, J. D., Mattson, C. A., and Ferguson, S. M., 2016, "Evaluation of System Evolvability Based on Usable Excess," J. Mech. Des., **138**(9), p. 091101.

[51] Watson, J. D., Allen, J. D., Mattson, C. A., and Ferguson, S. M., 2016, "Optimization of Excess System Capability for Increased Evolvability," Struct. Multidiscip. Optim., **53**(6), pp. 1277–1294.

[52] Cansler, E. Z., White, S. B., Ferguson, S. M., and Mattson, C. A., 2016, "Excess Identification and Mapping in Engineered Systems," J. Mech. Des., **138**(8), p. 081103.

[53] White, S., and Ferguson, S., 2017, "Exploring Architecture Selection and System Evolvability," *Proceedings of the ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 1–16.

[54] Elward, B., 2012, *The Boeing F/A-18E/F Super Hornet & EA-18G Growler: A Develompental and Operational History*, Atglen: Schiffer Publishing Ltd.

[55] Yenne, B., 2012, *B-52 Stratofortress: The Complete History of the World's Longest Serving and Best Known Bomber*, MBI Publishing Company.

[56] Davies, S., 2013, *Boeing B-52 Stratofortress: 1952 Onwards*, Haynes Publishing UK.

[57] Dorr, R., 2000, *B-52 Stratofortress*, Osprey Publishing.

[58] Knaack, M., 1988, *Encyclopedia of U.S. Air Force Aircraft and Missile Systems Volume 2 Post-World War II Bombers 1945-1973*, Office of Air Force Hisotry, Washington DC.

[59] “B-52H Stratofortress > U.S. Air Force > Fact Sheet Display” [Online]. Available: https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104465/b-52-stratofortress/. [Accessed: 25-Oct-2019].

[60] “B-1B Lancer > U.S. Air Force > Fact Sheet Display” [Online]. Available: https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104500/b-1b-lancer/. [Accessed: 25-Oct-2019].

[61] “B-2 Spirit > U.S. Air Force > Fact Sheet Display” [Online]. Available: https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104482/b-2-spirit/. [Accessed: 25-Oct-2019].

[62] Bowman, M., 2012, *Stratofortress: The Story of the B-52*, Casemate Publishers.

[63] Hawthorne-Tagg, L. S., 2004, *Development of the B-52: The Wright Field Story*, Air Force Material Command.

[64] Air Force Global Strike Command Public Affairs Office, 2015, “B-52 Stratofortress Fact Sheet” [Online]. Available: http://www.af.mil/AboutUs/FactSheets/Display/tabid/224/Article/104465/b-52-stratofortress.aspx. [Accessed: 01-Mar-2017].

[65] Miller, M. A., 2013, *U.S. Air Force Bomber Sustainment and Modernization: Background and Issues for Congress*.

[66] Department of the Air Force, 1999, *U.S. Air Force White Paper on Long Range Bombers*.

[67] Sanchez, M. V., 2011, “Venerable, Valued Bombers,” AIR FORCE Mag., pp. 28–33.

[68] Harris, K., 2005, “‘Old Warrior’ B-52 Gets Call in Afghanistan,” Stars Stripes.

[69] Gertler, J., 2014, *U.S. Air Force Bomber Sustainment and Modernization: Background and Issues for Congress*.

[70] Lorell, M., and Lavaux, H., 1998, *The Cutting Edge: A Half Century of U.S. Fighter Aircraft R&D*, RAND Corporation.

[71] Stevenson, J. P., 1993, *Pentagon Paradox: The Development of the F-18 Hornet*, Naval Institute Press.

[72] Jenkins, D. R., 2000, *F/A-18 Hornet: A Navy Success Story*, Mcgraw-Hill.

[73] US Comptroller General, 1980, *F/A-18 Naval Strike Fighter: Its Effectiveness Is Uncertain*, Washington DC.

[74] Young, J., Anderson, R., and Yurkovich, R., 1998, "A Description of the F/A-18E/F Design and Design Process," *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, Reston, Virigina.

[75] Eden, P., 2006, *Encyclopedia of Modern Military Aircraft*, Amber Books.

[76] US General Accounting Office, 1996, *NAVY AVIATION: F/A-18E/F Will Provide Marginal Operational Improvement at High Cost.*, Washington DC.

[77] U.S. Navy, 2008, "NATOPS Flight Manual Navy Model F/A-18A/B/C/D 161353 and Up Aircraft," p. 902.

[78] U.S. Navy, 2008, "NATOPS Flight Manual Navy Model F/A-18E/F 165533 and Up Aircraft," p. 934.

[79] Bolkom, C., 2006, *Navy F/A-18E/F Super Hornet and EA-18G Growler Aircraft: Background and Issues for Congress*.

[80] U.S. Navy, 2001, "Tactical Manual Pocket Guide: F/A-18 A/B/C/D Aircraft," p. 151.

[81] Wieringa, J. A., 2003, "Spiral Development and the F/A-18: Parallels from the Past Emerge in Spiral Development of the F/A-18A through F Varients," Progr. Manag., pp.

50–52.

[82] Faculty, T. A., Lim, D., and Fulfillment, I. P., 2009, “A Systematic Approach To Design for Lifelong Aircraft Evolution a Systematic Approach To Design for Lifelong,” Georgia Institute of Technology.

[83] U.S. Air Force, 2007, “National Museum of the US Air Force” [Online]. Available: http://www.nationalmuseum.af.mil/factsheets/factsheet.asp?id=384. [Accessed: 25-Mar-2014].

[84] Yadav, D., Long, D., Morkos, B., and Ferguson, S., 2019, “Estimating the Value of Excess: A Case Study of Gaming Computers, Consoles and the Video Game Industry,” *ASME International Design Engineering Technical Conference,* Anaheim, CA.

[85] Hale, J. L., 2017, “When Is It Time to Buy a New Computer? These Are the Tell-Tale Signs You Need to Say Goodbye,” Bustle.

[86] Shah, A., 2016, “The PC Upgrade Cycle Slows to Every Five to Six Years, Intel’s CEO Says,” IDG News Serv.

[87] Kaif, H., “Understanding What Is Inside Your Computer and How It Works – The 8 Am Edition” [Online]. Available: https://sites.jmu.edu/103oconnor-16/introduction-to-basic-computer-components-and-functions/. [Accessed: 30-Nov-2019].

[88] N/A, 2008, “Hard Stuff Trinity,” PC Gamer Mag., p. 79.

[89] Dingman, H., 2017, “9 Reasons Why PC Gaming Is Better than Consoles,” PC World [Online]. Available: https://www.pcworld.com/article/3118250/9-reasons-why-pc-gaming-is-a-better-value-than-consoles.html. [Accessed: 31-Mar-2020].

[90] Linneman, J., 2019, “Control on Console Is Brilliant - as Long as You Play on the Right Hardware,” Eurogamer [Online]. Available:

https://www.eurogamer.net/articles/digitalfoundry-2019-control-console-face-off. [Accessed: 31-Mar-2020].

[91] Bailey, K., 2019, “Games Are Running Worse Than Ever on the Base PS4 and Xbox One. Is There a Solution?,” USGamer [Online]. Available: https://www.usgamer.net/articles/games-are-worse-than-ever-on-the-base-ps4-and-xbox-one-is-there-a-solution. [Accessed: 31-Mar-2020].

[92] Zahran, M., *Graphics Processing Units (GPUs): Architecture and Programming Lecture 7: GPU Performance*.

[93] Keeney, R. L., 1973, “DECISION ANALYSIS WITH MULTIPLE OBJECTIVES: THE MEXICO CITY AIRPORT.,” Bell J Econ Manag. Sci, **4**(1), pp. 101–117.

[94] Chidamber, S. und Kon, H., 1994, “A Research Retrospective of Innovation Inception and Success: The Technology-Push Demand-Pull Question,” Int. J. Technol. Manag., **53**(9), pp. 1689–1699.

[95] Collopy, P. D., and Hollingsworth, P. M., 2011, “Value-Driven Design,” J. Aircr., **48**(3), pp. 749–759.

[96] Dupre, R., 2020, “Can GameStop Hang On Until the Console Upgrade Cycle Reboots?,” Nasdaq [Online]. Available: https://www.nasdaq.com/articles/can-gamestop-hang-on-until-the-console-upgrade-cycle-reboots-2020-01-19. [Accessed: 31-Mar-2020].

[97] O’Malley, J., 2016, “Played Right, Xbox One’s Upgrade Plan Could Win It the Console War,” TechRadar [Online]. Available: https://www.techradar.com/news/gaming/consoles/played-right-xbox-one-s-upgrade-plan-could-win-it-the-console-war-1316190. [Accessed: 31-Mar-2020].

[98] Thier, D., 2020, “The Xbox One And PS4 Were The Last Console Generation,” Forbes

[Online]. Available: https://www.forbes.com/sites/davidthier/2020/01/11/the-xbox-one-and-ps4-were-the-last-console-generation/#3f41a5863db2. [Accessed: 31-Mar-2020].

[99] Gilbert, B., 2020, "How Microsoft's next Xbox Plans to End the Console Wars with Sony," Bus. Insid. [Online]. Available: https://www.businessinsider.com/playstation-xbox-console-wars-end-microsoft-sony-2020-1#3-going-forward-its-just-xbox-3. [Accessed: 31-Mar-2020].

[100] Upton, D. M., 1995, "What Really Makes Factories Flexible?," Harv. Bus. Rev., **73**(4), pp. 74–84.

[101] Pasqual, M. C., and De Weck, O. L., 2012, "Multilayer Network Model for Analysis and Management of Change Propagation," Res. Eng. Des., **23**(4), pp. 305–328.

[102] Belt, A., 2013, "Exploring Customization Option Assessment and Selection during the Early Stages of Redesign," North Carolina State University.

[103] Hamraz, B., Caldwell, N. H. M., and John Clarkson, P., 2012, "A Multidomain Engineering Change Propagation Model to Support Uncertainty Reduction and Risk Management in Design," J. Mech. Des., **134**(10), p. 100905.

[104] Hamraz, B., Caldwell, N. H., and Clarkson, P. J., 2013, "A Matrix-Calculation-Based Algorithm for Numerical Change Propagation Analysis," IEEE Trans. Eng. Manag., **60**(1), pp. 186–198.

[105] Sosa, M., Mihm, J., and Browning, T. R., 2011, "Degree Distribution and Quality in Complex Engineered Systems," J. Mech. Des., **133**(10), p. 101008.

[106] Albert, R., Jeong, H., and Barabási, A.-L., 2000, "Error and Attack Tolerance of Complex Networks," Nature, **406**(6794), pp. 378–382.

[107] Strogatz, S. H., 2001, "Exploring Complex Networks," Nature, **410**(6825), pp. 268–276.

[108] Barabási, A.-L., and Albert, R., 1999, “Emergence of Scaling in Random Networks,” Science (80-.)., **286**, pp. 509–512.

[109] Mehrpouyan, H., Haley, B., Dong, A., Tumer, I. Y., and Hoyle, C., 2015, “Resiliency Analysis for Complex Engineered System Design,” Artifcial Intell. Eng. Des. Anal. Manuf., **29**(1), pp. 93–108.

[110] Newman, M. E. J., *The Mathematics of Networks*, Ann Arbor.

[111] Barber, P., Buxton, I., Stephenson, H., and Ritchey, I., 1999, “Design for Upgradeability: Extending the Life of Large Made to Order Products at the Design Stage,” Int. Prod. Dev. Manag. Conf., pp. 75–85.

[112] Saleh, J. H., Hastings, D. E., and Newman, D. J., “Spacecraft Design Lifetime,” J. Spacecr. Rockets, **39**(2).

[113] Brathwaite, J., and Saleh, J. H., “Beyond Cost and Performance, a Value-Centric Framework and Pareto Optimization for Communication Satellites *.”

[114] Saleh, J. H., Jordan, N. C., and Newman, D. J., 2007, “Shifting the Emphasis: From Cost Models to Satellite Utility or Revenue Models. The Case for a Value-Centric Mindset in Space System Design,” Acta Astronaut., **61**(10), pp. 889–900.

[115] Geng, F., Dubos, G. F., and Saleh, J. H., 2016, “Spacecraft Obsolescence: Modeling, Value Analysis, and Implications for Design and Acquisition,” IEEE Aerosp. Conf. Proc., **2016-June**, pp. 1–13.

[116] Ryan, E. T., Jacques, D. R., and Colombi, J. M., 2013, “An Ontological Framework for Clarifying Flexibility-Related Terminology via Literature Survey,” Syst. Eng., **16**(1), pp. 99–110.

[117] Ross, A. M., Rhodes, D. H., and Hastings, D. E., 2008, “Defining Changeability :

Reconciling Flexibility , Adaptability , Scalability , Modifiability , and Robustness for Maintaining System Lifecycle Value," Syst. Eng., **11**(3), pp. 246–262.

[118] Nembhard, H., and Aktan, M., 2010, *Real Options in Engineering Design, Operations, and Management*, CRC Press.

[119] Kalligeros, K., 2009, "Real Options in Engineering Design," *Real Options in Engineering Design, Operations, and Management*, pp. 127–153.

[120] Neufville, R. De, Asce, L. M., Scholtes, S., and Wang, T., 2005, "Real Options by Spreadsheet : Parking Garage Case Example Valuing Real Options by Spreadsheet : Parking Garage Case Example," J. Infrastruct. Syst., **12**(3), pp. 1–19.

[121] Delligatti, L., 2014, *SysML Distilled: A Brief Guide to the Systems Modeling Language*, Addison-Wesley.

[122] Tilstra, A. H., Seepersad, C. C., and Wood, K. L., 2009, "Analysis of Product Flexibility for Future Evolution Based on Design Guidelinesand a High-Definition Design Structure Matrix," *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2009), August 30–September 2, 2009 , San Diego, California, USA*, ASME, pp. 951–964.

[123] Ross, A. M., and Rhodes, D. H., 2008, "Using Natural Value-Centric Time Scales for Conceptualizing System Timelines through Epoch-Era Analysis," *18th Annual International Symposium of the International Council on Systems Engineering, INCOSE 2008*, pp. 2250–2264.

[124] Yadav, D., Long, D., Morkos, B., and Ferguson, S. M., 2019, "ESTIMATING THE VALUE OF EXCESS: A CASE STUDY OF GAMING COMPUTERS, CONSOLES AND THE VIDEO GAME INDUSTRY," *Proceedings of ASME 2019 IDETC/CIE*, p. 14.

[125] PC Gamer, 2010, “Hard Stuff Trinity,” PC Gamer.

[126] Gorod, A., Sauser, B., and Boardman, J., 2008, “System-of-Systems Engineering Management: A Review of Modern History and a Path Forward,” IEEE Syst. J., **2**(4), pp. 484–499.

[127] Long, D., and Ferguson, S. M., 2019, “An Excess Based Approach to Change Propagation,” *Proceedings - IDETC 2019*.

[128] Sutton, R. S., and Barto, A. G., 2018, *Reinforcement Learning: An Introduction*.

APPENDICES

Appendix 1: Remaining System Direct Likelihood DSMs

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1		0.7	0.7	0.2	0.2		0.2	0.2	0.5		0.2	0.7	0.2	0.7	0.5	0.7	0.7	0.2	0.7
2	0.2																		
3										0.7								0.2	
4			0.5			0.2	0.2	0.2			0.2				0.2				
5						0.2												0.2	
6				0.2	0.2						0.2				0.2			0.2	
7				0.2	0.2						0.2							0.2	
8	0.2	0.2		0.2			0.2				0.2				0.2				
9	0.2	0.2	0.7							0.2									
10			0.7								0.2								
11			0.2	0.2	0.2	0.2	0.2	0.2		0.2		0.2			0.2				
12		0.5	0.7	0.2	0.5			0.2	0.2	0.5	0.5				0.5		0.2	0.2	0.2
13	0.2	0.5		0.2		0.2						0.2						0.2	0.2
14	0.5	0.5	0.7	0.5				0.5	0.5	0.2	0.5	0.7	0.7		0.5		0.7	0.5	0.5
15			0.2													0.2			
16	0.2	0.2						0.2	0.2										0.2
17	0.2		0.2						0.2										
18			0.5	0.2	0.2		0.2												
19	0.2	0.2	0.2							0.2		0.2				0.2			

Figure A1.1: Helicopter [104]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1		0.5	0.5	0.5			0.5	0.2			0.2	0.8	0.2	
2	0.5					0.5								
3	0.5			0.5										
4	0.5		0.5											
5						0.5					0.2		0.5	
6		0.5			0.5		0.2			0.2				
7	0.5					0.2		0.2	0.5	0.8	0.2			
8	0.2						0.2				0.2			
9							0.5				0.2			
10						0.2	0.8							
11	0.2				0.2		0.2	0.2	0.2					
12	0.8													
13	0.2				0.5									0.5
14													0.5	

Figure A1.2: UGV [36]

	1	2	3	4	5	6	7	8	9
1		0.3		0.4			0.2		
2	0.4						0.3		
3					0.2			0.8	0.6
4	0.1				0.4		0.4		
5			0.2	0.2			0.4	0.4	0.8
6							0.6		0.8
7	0.2			0.3	0.3	0.6			
8			0.2		0.2				0.6
9			0.4		0.5	0.8		0.6	

Figure A1.3: Fan [102]

Appendix 2: System CPC by Generation Figures

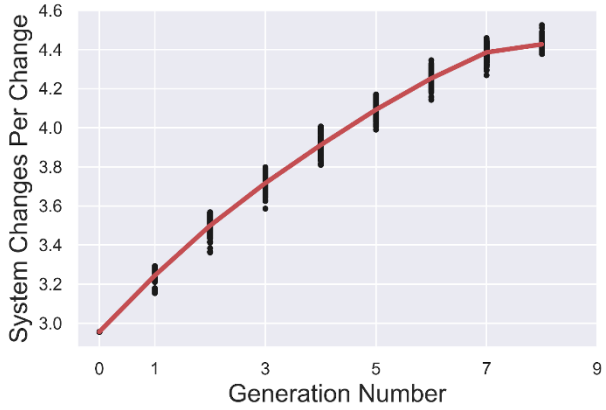


Figure A2.1: Hair Dryer

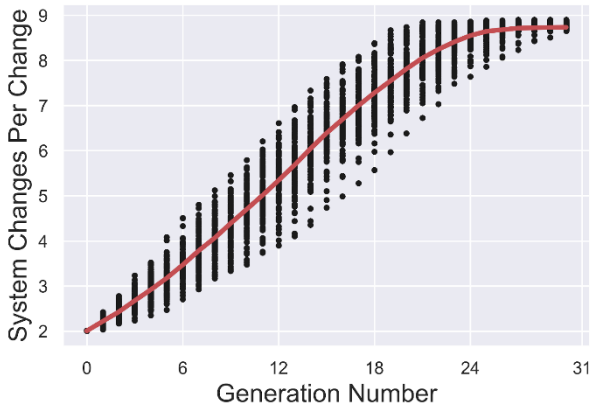


Figure A2.2: UGV

Appendix 3: Component CPC Figures

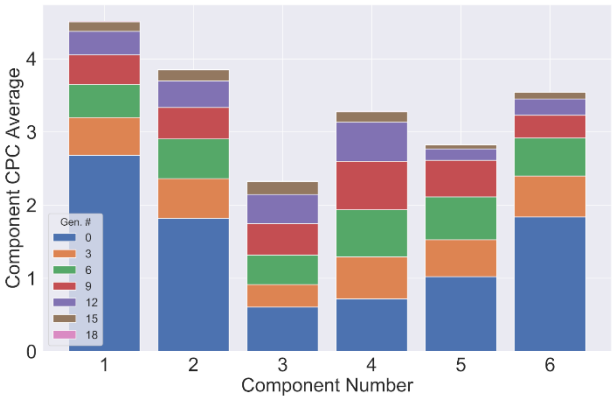


Figure A3.1: Grill

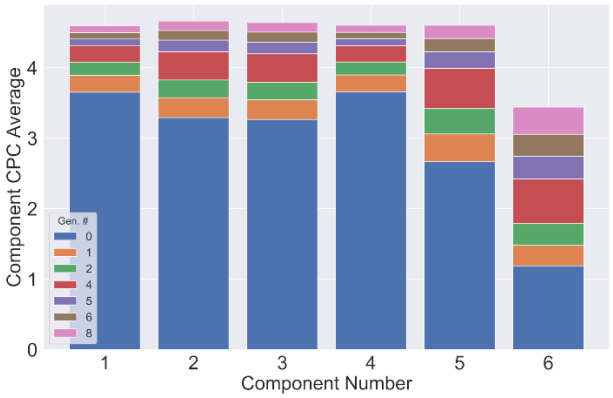


Figure A3.2: Hair Dryer

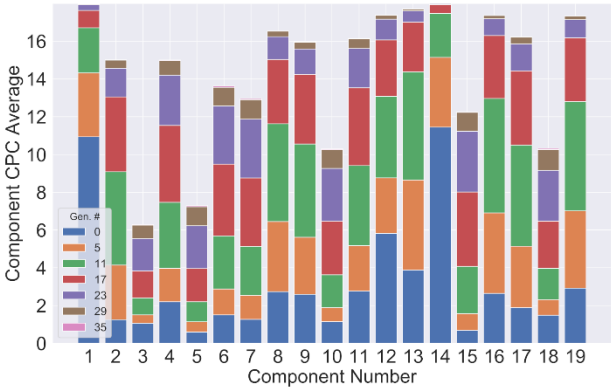


Figure A3.3: Helicopter

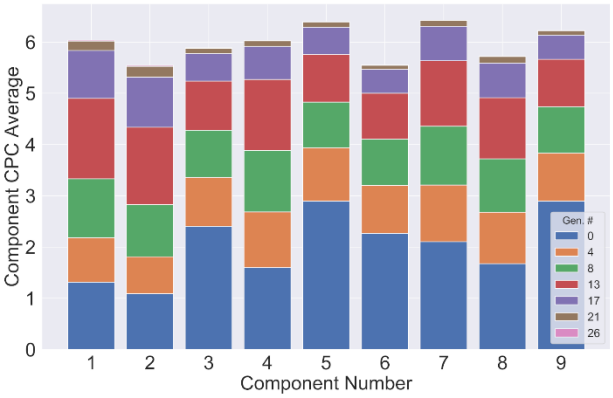


Figure A3.4: Fan

Appendix 4: Component Rank Figures

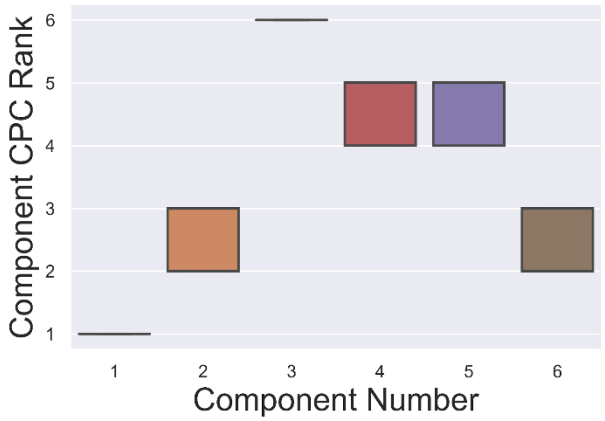


Figure A4.1: Grill

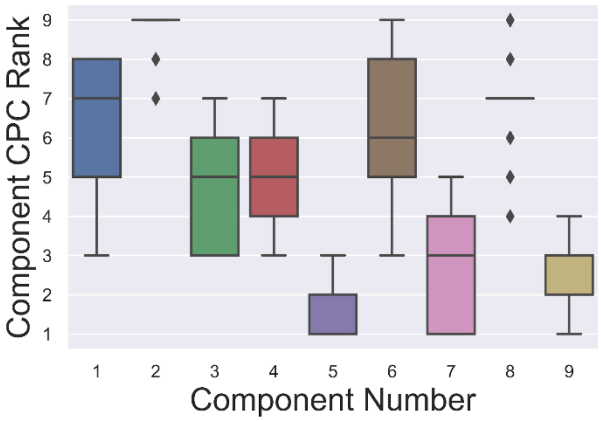


Figure A4.2: Fan

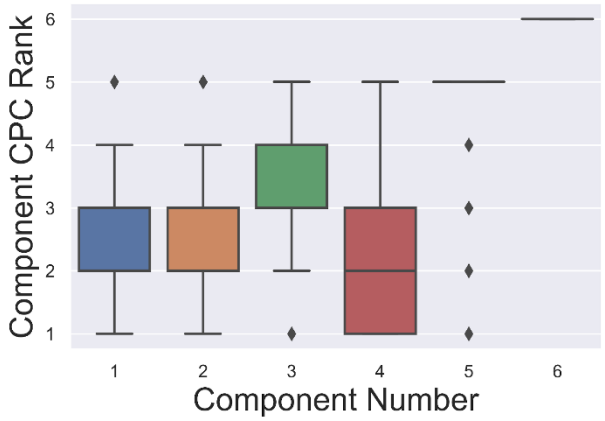


Figure A4.3: Hair Dryer

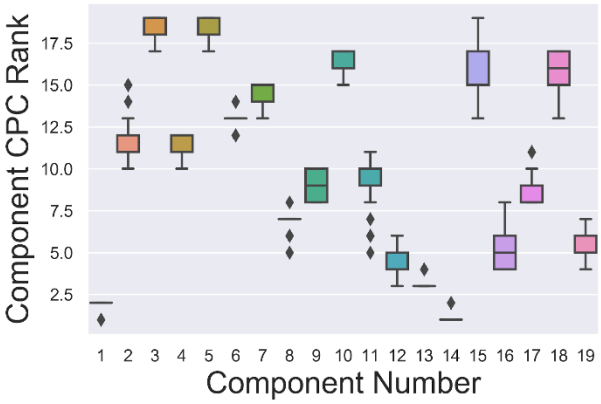


Figure A4.4: Helicopter