ABSTRACT

MORRISON, PATRICK JAMES. A Security Practices Evaluation Framework. (Under the direction of Laurie Williams.)

Motivation. Security breaches, and the software vulnerabilities that enable them, have increasingly become headline news. Avoiding or resolving vulnerabilities during software development has become a consideration for many organizations that build and ship software.

Research Problem. Many software development security practices have been recommended. However, empirical evidence on the application and results of the practices is limited. The goal of this research is to support researcher and practitioner adoption of security practices by developing a model for how security practice adherence affects security outcomes and a framework for the collection of software development context factors, practice adherence, and security outcomes.

Approach. To support the collection of empirical evidence for the effects of security practice use in software development, we propose a model, the Security Outcomes Theoretical Model (SOTM), for how security practice adherence affects security outcomes, supported by a measurement framework, the Security Practices Evaluation Framework (SP-EF). SOTM is a set of constructs and relationships that embody a theory of how security practices affect security outcomes. SP-EF is a measurement framework that includes repeatable instructions for identifying the use of these practices, recording the context of the team using the practices, and measuring the security outcomes for the software.

Results. We conducted a literature review to identify software development security context factors, security practices, and outcome measures. We identified a set of security practices, practice adherence metrics, outcome measures, and context factors, and assembled them as a measurement framework, SP-EF. We conducted a case study in which we collected all SP-EF data from an industrial software development project. We found agreement between the researcher and team views of security practice use on the project, and evaluated the effectiveness of automated means of assessing practice adherence. We identified use of all of the practices

specified in SP-EF by one or more survey participants. We conducted a survey of open source development projects to assess the use of the SP-EF security practices and the degree to which our adherence measures correlate with security practice use. We found empirical support for the use of the SP-EF security practices. We found that Training has a positive, statistically significant correlation with Usage, suggesting that investment in training supports practice usage. Finally, we assessed whether the theorized relationships in SOTM hold in observational data by combining SP-EF measurements available in published datasets. Our data suggest that assessing the state of software security requires accounting for both software development context factors and the software usage context factors.

Contributions. Our contributions include:

- The Security Outcomes Theoretical Model, a proposed model of the constructs affecting software development security outcomes;
- SP-EF, a proposed set of measures for assessing security in the software development process including instructions for data collection, and
- Empirical evaluation of the proposed model and metrics using two open source datasets.

© Copyright 2017 by Patrick James Morrison

All Rights Reserved

A Security Practices Evaluation Framework

by
Patrick James Morrison

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

Computer Science

Raleigh, North Carolina

2017

APPROVED BY:

Chris Parnin

Emerson Murphy-Hill

Johnathan Stallings

Laurie Williams
Chair of Advisory Committee

DEDICATION

Dedicated to my wife Karen, who persuaded me to begin
and who's been beside me every step of the way.

BIOGRAPHY

Patrick Morrison was born in Avon Park, Fl. He graduated from Flagler Palm Coast High School, where he was part of the National Honor Society and the Chess Club. He graduated from the University of Florida with a Bachelor of Science in Computer Science in 1986. He spent time in a sequence of development jobs for large corporations, and large and small consulting firms, leading to teaching a class in web development at Florida Atlantic University. One thing lead to another, and he earned a Master of Science in Computer Science from Florida Atlantic University in 2007. One more thing led to another, and he returned to school full-time to pursue a Doctorate in Philosophy at North Carolina State University. Currently, Patrick is working at IBM.

ACKNOWLEDGEMENTS

I am grateful to my advisor, Laurie Williams, for the time and effort she took in introducing me to software engineering research, for all of the patience, encouragement, funding, and feedback that came with the introduction, and for the vision and willingness behind it all. Thank you to my committee members, Emerson Murphy-Hill, Chris Parnin, and Jonathan Stallings, for all of the time spent, and all of the advice and support you've so generously given.

I am greatly indebted to the members of Laurie's research group: Andrew Austin, John Slankas, Pat Francis, Eric Helms, JeeHyun Hwang, Da Young Lee, Jason King, Andy Meneely, Rahul Pandita, Akond Rahman, Maria Riaz, Yonghee Shin, Ben Smith, Will Snipes, Chris Theisen, Akond Rahman, and Sarah Elder. From critiquing my work helpfully, to various project collaborations, to sharing the woes and joys of life inside and outside research, I am grateful. Many of my new skills are your old skills, and I trust that our new friendships will become old friendships over time. Much the same can be said for the many brilliant, creative, and dedicated denizens of the software engineering labs in rooms 3228 and 3231, e.g. Aaron Massey, Brittany Johnson, Adam Marrs (honorary denizen), Jim Witschey, Titus Barik, Kevin Lubick, Michael Winters, Justin Smith, and more.

I am grateful to teachers, mentors, and friends who have inspired, encouraged, taught, admonished, and inspired me. Whether through a kind word, a new perspective, or an old reminder, your influence on me is invaluable and cherished. The list can only be partial, but it includes Mrs. Chase, James Daly, Henly Wolin, Sylvia Brady, Ron Conkling, Charles Pike, Kevin Desruisseau, Bruce Edwards, Patrick Forney, Phil Spadaro, Morgan Baker, Eduardo Fernandez, Michael Van Hilst, Shihong Huang, Maria Petrie, Jerry Weinberg, Linda Rising, Rob DeLine, Brendan Murphy, Kim Herzig, Eric Franklin, Russell Teague, and Andy Rindos.

I am grateful to so many in the Triangle Church of Christ who have shared their lives and hearts with us, around dinner tables, prayer circles, service activities, and worship services. I'm especially grateful for those who have watched over Daniel when Karen or I had to be elsewhere: Kris Bailey, Dorothy, Ana, and Andrew Felsher, the Hoegemeyer clan, Joy Morris, Casey Kuan, Alec and Kaley Nunez, Cari Overturf, the Teague clan. I am grateful to Cub Scout Pack 222 for taking Daniel and I in, and adding camping, woodworking, Pinewood Derby racing, and dozens of other cool things to Daniel's childhood. We are grateful to the staffs of Wiley Elementary School and the Alexander Family YMCA for taking Daniel into their hearts as well as their classrooms.

I am grateful to the engineers behind my Pomodoro kitchen timer, Ikea lamp, Acura Legend, Apple Macbook, the R Project, and the 'Hadleyverse' for making things that helped with research and with life. I'm especially grateful to all of those working on open source software

for the differences your contributions make for me and for so many others. I am grateful for a thousand and one conversations supported by coffee at CupAJoe, beer at Mitch's, and meals at Noodles and Company. I am grateful for Chet Atkins, Johann Sebastian Bach, David Byrne, Guy Clark, Ry Cooder, Miles Davis, Brian Eno, Ella Fitzgerald, David Gilmour, Andy Getch, Nanci Griffith, Mark Knopfler, Bob Marley, Joe Pass, John Prine, Paul Roub, John Santa, Ronnie Van Zant, Paul Westerberg, Doc Watson, Dar Williams, Stevie Wonder, and all the other singers, songwriters, and musicians who kept me company over long stretches of writing and contemplation.

The family I came from supplied me with abundant love, creativity, and vision in the form of Leo and Liilia Morrison, Michael, Kathleen, and James, and Karl and Leida Talts.

The family I helped create sustains me in so many ways. Daniel brings great joy into our lives, and Karen brings joy as well as just about everything else. I'm pretty sure I wouldn't even be able to type these lines if not for a thousand things she's done to keep things running while I type.

Most of all, I am grateful to be able to pursue the Author of all things, and to know that He wants to know me as well.

This work has been supported, in part, by an IBM PhD Fellowship, and by the USA National Security Agency (NSA) Science of Security Lablet at NCSU. Any opinions expressed in this report are those of the author(s) and do not necessarily reflect the views of IBM or the NSA. We are grateful to the survey participants for their time, effort, and contributions.

Think - IBM

Think and Do - NCSU

Esse quam videri - North Carolina

Mercy triumphs over judgment. - James 2:13

TABLE OF CONTENTS

LIST OF TABLES	xi
LIST OF FIGURES	xii
Chapter 1 Introduction	1
1.1 Solution Methodology	3
1.1.1 Security Outcomes Theoretical Model	3
1.1.2 The Security Practices Evaluation Framework	4
1.2 Thesis statement	4
1.3 Research Questions and Associated Studies	4
1.4 Contributions	5
Chapter 2 Background and Related Work	7
2.1 Definitions	7
2.2 Measurement Frameworks	8
2.3 Structural Equation Modeling	9
2.4 Software Development Security Metrics	12
2.5 Software Development Security Practices	13
2.5.1 The Building Security In Maturity Model	13
2.5.2 The Microsoft Security Development Lifecycle	14
2.5.3 SAFECode	14
2.5.4 OWASP CLASP	14
2.5.5 Other Security Practice Lists and Surveys	15
2.6 Technology Adoption	16
2.7 Practice Adherence	17
Chapter 3 Security Metrics and Practices Mapping Study	18
3.1 Introduction	18
3.1.1 Security Metrics Systematic Literature Review	18
3.1.2 Security Practices Review	19
3.2 Methodology	20
3.2.1 Search Strategy	20
3.3 Results	26
3.3.1 RQ1.1: What software lifecycle security metrics have been proposed in the literature?	26
3.3.2 RQ1.2: What are the people, artifacts, and process elements being measured by software lifecycle security metrics?	38
3.3.3 RQ1.3: What validation approaches are used for software lifecycle security metrics?	39
3.3.4 RQ1.4 During what phases of the software development lifecycle are software lifecycle security metrics measured?	39
3.4 Discussion	41
3.4.1 Themes in Security Metrics	41

3.4.2	What To Measure?	44
3.4.3	Fundamental Security Questions, revisited	44
3.5	Limitations	46
3.6	Contributions of the Systematic Mapping Study	47
3.7	RQ1.5: What software development security practices are used by software development teams?	47
3.7.1	Methodology	48
3.7.2	Practice classification results	49
3.7.3	Discussion	51
3.7.4	Limitations	51
Chapter 4	A Security Practices Evaluation Framework	53
4.1	Introduction	53
4.2	Context Factors	55
4.3	Practice Adherence	55
4.3.1	Outcome Measures	56
4.4	The Security Outcomes Theoretical Model (SOTM)	56
4.4.1	Structural Model Constructs	57
4.4.2	Structural Model Relationships	58
4.4.3	Measurement Model: Metrics, Variables, and Relationships	61
4.5	Contribution	63
Chapter 5	Measuring Security Practice Use in Software Development	64
5.1	Methodology	65
5.2	Industrial Case Study Results	65
5.2.1	RQ2.1: Can the complete set of SP-EF measurements be collected for a software development project?	66
5.3	Discussion	73
5.4	Lessons Learned	74
5.5	Limitations	75
5.6	Contributions of the Measurement Case Study	75
Chapter 6	Surveying Security Practice Use in Software Development	77
6.1	Introduction	77
6.2	Methodology	78
6.2.1	Study Methodology	78
6.2.2	RQ2.2: What software development security practices are used by software development teams?	78
6.2.3	RQ2.3: Does security practice adherence, as measured by Ease of use, Effectiveness, and Training, correlate with software development security practice use?	79
6.3	Results	80
6.3.1	Practice classification results	80
6.3.2	Survey Subjects and Responses	80
6.3.3	Research Question Results	81

6.4	Discussion	85
6.5	Limitations	88
6.6	Contributions of the Survey Study	88
Chapter 7 Quantifying Security Context Factors in Software Development . .		90
7.1	Expressing SOTM as a SEM Model	91
7.2	Case Study Methodology	93
7.2.1	Step 1: Select case study subject	93
7.2.2	Step 2: Link data source variables to the model constructs	93
7.2.3	Step 3: Evaluate collected data for fit problems	94
7.2.4	Step 4: Estimate SEM model	94
7.2.5	Step 5: Test model fit	95
7.2.6	Step 6: Perform Re-specification, if necessary	95
7.2.7	Step 7: Report Results	96
7.3	Case Study	97
7.3.1	Data selection	98
7.3.2	Data collection	99
7.3.3	Estimation	99
7.3.4	Model Fit	100
7.3.5	Re-specification	101
7.3.6	Reporting Results	102
7.3.7	Discussion	104
7.4	Limitations	106
7.5	Contributions of the Quantification Study	107
Chapter 8 Case Studies using SP-EF		109
8.1	Methodology	109
8.1.1	Subject selection	110
8.1.2	Data Collection	110
8.1.3	Research question analysis	111
8.2	Bitcoin Case Study	112
8.2.1	Data Collection	112
8.2.2	Results	113
8.3	IBM wasCloud Case Study	116
8.4	phpMyAdmin Case Study	119
8.4.1	Data Collection	119
8.4.2	Results	120
8.5	OpenSSL Case Study	123
8.5.1	Data Collection	126
8.5.2	Results	126
8.6	Research Question Answers	129
8.6.1	RQ2.1: Can the complete set of SP-EF measurements be collected for a software development project?	129
8.6.2	RQ3: How does security practice adherence affect software development security outcomes?	131

8.7	Discussion	134
8.8	Limitations	135
8.9	Conclusion	135
Chapter 9	Conclusion	136
9.1	Contributions	137
9.2	Future Work	138
BIBLIOGRAPHY		138
APPENDICES		147
Appendix A	Selected Papers	148
Appendix B	Survey Questionnaire	153
Appendix C	SP-EF Guidebook	155
C.1	SP-EF Measurement Guidebook	155
C.1.1	Abstract	155
C.2	Introduction	156
C.2.1	Domain	157
C.2.2	Context Factors	158
C.2.3	Language	158
C.2.4	Confidentiality, Integrity, and Availability Requirements	158
C.2.5	Methodology	159
C.2.6	Source Code Availability	159
C.2.7	Team Size	160
C.2.8	Team Location	161
C.2.9	Operating System	161
C.2.10	Product Age	162
C.2.11	Number of Identities	162
C.2.12	Number of Machines	163
C.2.13	Source Lines of Code (SLOC)	163
C.2.14	Churn	164
C.2.15	Perform Security Training	166
C.2.16	Description	166
C.2.17	Apply Data Classification Scheme	166
C.2.18	Apply Security Requirements	167
C.2.19	Apply Threat Modeling	168
C.2.20	Document Technical Stack	169
C.2.21	Apply Secure Coding Standards	170
C.2.22	Apply Security Tooling	171
C.2.23	Perform Security Review	172
C.2.24	Perform Security Testing	173
C.2.25	Description	174
C.2.26	Links	174
C.2.27	Publish Operations Guide	174
C.2.28	Perform Penetration Testing	175

C.2.29 Track Vulnerabilities	176
C.2.30 Improve Development Process	177
C.2.31 Subjective Practice Adherence Measurement	178
C.2.32 Objective Practice Adherence Measurement	179
C.2.33 Per Vulnerability Attributes	180
C.2.34 Pre-Release Defects	180
C.2.35 Post-Release Defects	181
C.2.36 Vulnerability Density	181
C.2.37 Pre-Release Vulnerabilities	182
C.2.38 Post-Release Vulnerabilities	182
C.2.39 Vulnerability Removal Effectiveness	183
C.2.40 Data Collection	183

LIST OF TABLES

Table 2.1	SEM Notation, adapted from Hildreth [27]	10
Table 3.1	Research Question Keywords	21
Table 3.2	Keyword Synonyms	21
Table 3.3	Search Results	24
Table 3.4	Security Metric Categories and Subcategories	27
Table 3.5	Metric Ids by Phase and Subject	38
Table 3.6	Studies by Evaluation Technique	39
Table 3.7	Studies by Lifecycle Phase	40
Table 3.8	Lifecycle Phase by Metric Category	41
Table 3.9	Metrics referenced in two or more papers and evaluated in an industrial context	45
Table 3.10	Security Practices By Source	50
Table 3.11	SP-EF Security Practice Descriptions	52
Table 4.1	Model Context Factors, Measures, and Hypotheses	62
Table 5.1	Practice Adherence Comparison Table	68
Table 6.1	Survey Invitations and Responses, by Project	81
Table 6.2	Practice Adherence Metrics Usage	87
Table 7.1	SOTM Lavaan Syntax	92
Table 7.2	Combined Demographics for 697 Projects	100
Table 7.3	Global Fit Measures and Results	101
Table 7.4	OpenHub-NVD Respecified Model Results	102
Table 7.5	OpenHub-NVD Respecified Model Residuals	104
Table 8.1	Bitcoin Manual Review Evidence	114
Table 8.2	phpMyAdmin Manual Review Evidence	121
Table 8.3	SP-EF Model Context Factors, Measures, and Case Study Results	125
Table 8.4	OpenSSL Manual Review Evidence	127
Table A.1	Selected Papers	149
Table A.2	Selected Papers	150
Table A.3	Selected Papers	151
Table A.4	Selected Papers	152

LIST OF FIGURES

Figure 4.1	Structural Model Overview	60
Figure 4.2	Graphical display of lavaan syntax for DevelopmentRisk $\approx$ Theorized measurement variables from Table 4.1	61
Figure 5.1	Context Factors	67
Figure 5.2	Survey Results	72
Figure 6.1	Security Practice Adherence Measurement Model	80
Figure 6.2	Security Practices Usage Frequency	82
Figure 7.1	Respecified OpenHub-NVD Combined Model	103
Figure 8.1	Bitcoin Metrics 2010-2016	117
Figure 8.2	wasCloud Metrics 20103-2015	118
Figure 8.3	phpMyAdmin Metrics 2001-2014	124
Figure 8.4	OpenSSL Metrics 2013-2016	130
Figure 8.5	Vulnerability Density (VDensity) and Vulnerability Removal Effectiveness (VRE) Boxplots for Case Study Projects	133

Chapter 1

Introduction

Vulnerabilities introduced during software development are one of the causes of security breaches in software systems. For example, In February 2017, security researchers discovered a programming bug in a HTML parser at Cloudflare¹. The parser was sending decrypted encryption keys and passwords along with ordinary web requests, resulting in unencrypted sensitive data being cached across the Internet. While the programming bug, now known as ‘Cloudbleed’², was on a single line of code, the bug had widespread effects because of the code’s context, running on many servers and sending sensitive results to many clients. The example of Cloudbleed suggests that software development teams must consider the software’s environment and data as well as the technical characteristics of their software when evaluating their software’s security.

The IEEE defines security [31] as ‘all aspects related to defining, achieving, and maintaining confidentiality, integrity, availability, non-repudiation, accountability, authenticity, and reliability of a system.’ Multiple software technical and process characteristics contribute to the potential for software vulnerabilities. For example, code size [1], code churn [86, 57], and language [78] have been correlated with vulnerabilities. Similarly, multiple software usage characteristics, for example, access to sensitive data³, management of financial information [26], and the presence of a piece of software on large numbers of machines⁴ have been correlated with vulnerabilities. Two pieces of software may have similar technical characteristics (e.g. same language, same size) but different security outcomes as a consequence of the uses to which the

¹<https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug/>

²<https://blog.cloudflare.com/quantifying-the-impact-of-cloudbleed>

³<http://heartbleed.com/>

⁴<http://www.cert.org/historical/advisories/CA-2001-19.cfm>

software is put. For example, between a game level editor and a database system of similar source code size, the database system managing credit card data may be more prone to attack and to have more publicly reported vulnerabilities ⁵, than the game level editor managing custom game levels.

As evidenced by Cloudblood, vulnerabilities due to even small mistakes during software development can have significant consequences. As one approach to reducing or eliminating vulnerabilities, software development teams are recognizing the need to ‘build security in’ [52] during the development process. Large companies have developed sophisticated approaches to applying security practices in software development. Example lists include the Building Security in Maturity Model [52] (BSIMM), the Microsoft Security Development Lifecycle [29] (SDL), the Software Assurance Forum for Excellence in Code [89] (SAFECode), and the Open Web Application Security Project (OWASP) Software Security Assurance Process [49] (SSAP). The BSIMM is a survey and inventory of security practices identified in large organizations. The authors of the BSIMM suggest that by clearly noting objectives and by tracking practices with metrics tailored to your own situation, you can use the BSIMM as a measurement tool. However, determining the objectives, tracking the practices, determining which metrics to collect, and organizing and analyzing the resulting data goes beyond the scope of the BSIMM. Indeed, the BSIMM explicitly avoids recommending practices and recommends organizations develop their own initiatives for selecting and evaluating practices. Further, practices and initiatives appropriate for a large, widely used product may be overly complicated or expensive for a small team. Many software projects, including key infrastructure like OpenSSL and NTP ⁶, are developed by small teams with limited resources.

An organization’s, team’s, or individual developer’s ability to measure is the first step in balancing the trade off between the costs in time, effort, and complexity of applying security practices and the achievement of an appropriate level of security in delivered software. In a survey of the literature [93], Verendel concluded that the foremost reason for the unknown validity of software security measurement is the lack of repeated large-sample empirical validation of the specific quantitative methods. Measurement frameworks [36, 98] offer a means for categorizing empirical studies and organizing them into a body of knowledge, supporting repeated large-sample empirical validation. Developing a framework for measuring the use and effect of security practices, applying and promoting it across projects, and supporting the collection and

⁵Following Krsul [40] and Ozment [73], we define a vulnerability as an instance of a mistake in the specification, development, or configuration of software such that its execution can violate the explicit or implicit security policy.

⁶<http://www.ntp.org/>

analysis of the results are all necessary components of meeting Verendels concern.

Whether at the team, organization, or industry level, a body of knowledge on the effects of security practice use in software development would support practitioners and researchers in project planning, security assurance, and process improvement. For example, one question in software development security practice selection is which practices should be adopted to help the team achieve its security goals. Team selection of security practices would be strengthened by empirical evidence for the effectiveness of applying security practices. A second question is whether the practices selected are worth the investment given the project goals. A team building a single-user application that does not store sensitive data may require relatively little in the way of security practices and security effort. A team building a system that manages high-value data may require a comprehensive set of security practices and high levels of security practice effort. Systematically determining where a software development project lies on this continuum of security need versus the assurance afforded by applying security practices requires a scheme for measuring how environmental characteristics, system characteristics and security practice use affect security need and protection. Answers for these questions based on empirical evidence would benefit software development teams. *The goal of this research is to support researcher and practitioner adoption of security practices through the development of a model for how security practice adherence affects security outcomes and a framework for the collection of software development context factors, practice adherence, and security outcomes.*

1.1 Solution Methodology

To support software development teams in assessing security practice effectiveness in the context of software development project, in this research a model for how security practice use affects security outcomes was developed, as well as a measurement framework for the collection of empirical data on security practices and outcomes.

1.1.1 Security Outcomes Theoretical Model

To support detailed analysis for how security practices affect security outcomes, we represent each element of the framework in a model of the software development lifecycle, enabling study of the relative importance of each security practice in software development, and measurement of the validity and reliability of the framework.

We develop a model hypothesizing relationships between software development context factors, software usage factors, adherence to security practices, and security outcomes. We hypoth-

esize that the four constructs are related as follows:

- **H1** Software usage factors are associated with negative Security Outcomes
- **H2** Software development context factors are associated with negative Security Outcomes
- **H3** Practice Adherence is associated with Software development context factors

1.1.2 The Security Practices Evaluation Framework

We propose the Security Practices Evaluation Framework (SP-EF), a measurement framework to enable empirical data collection for security practice use and outcomes in software development. SP-EF contains three categories of data elements: context factors, practice adherence metrics, and outcome measures. Context factors are a set of attributes and values that are used to provide a basis of comparison between projects measured using SP-EF. The practice adherence metrics are a set of attributes and values that are used to describe security practices in use on a project, and the degree to which each practice is adhered to by the project team. Outcome measures are a set of attributes and values that are used to describe the security-related outcomes of the project. We define data elements of SP-EF and provide guidance on how to collect the data elements in the form of an online guidebook [63]. The framework is described in Section 4 and published online [63]. To support SP-EF use, we have built reusable data collection infrastructure in the form of the guidebook, a survey questionnaire, and mining infrastructure. The SP-EF data is linked to the SOTM to support analysis of the effects of context factors and practice adherence on security outcomes.

1.2 Thesis statement

My thesis is as follows: *Measuring the effect of security practice adherence on software security outcomes requires accounting for software development context factors and software usage context factors.*

1.3 Research Questions and Associated Studies

To address our research goal, we address the thesis through posing three research questions:

- *RQ1* What measurements (context factors, security practices, adherence measures, and outcome measures) affect software development security outcomes? (Chapter 3)

- *RQ2* Can the measurements affecting software development security outcomes be measured on software development projects? (Chapter 5, Chapter 6, Chapter 8)
- *RQ3* How do context factors and security practice adherence affect software development security outcomes? (Chapter 7, Chapter 8)

To investigate these research questions, we conducted five studies:

1. *Security Metrics Mapping Study* [65] (RQ1): We conducted a systematic mapping study of software development security metrics. For each metric, we identify the subject being measured, how the metric has been validated, and how the metric is used. We categorized the metrics, and give examples of metrics for each category. We conducted a literature review of security practice lists, extracting 16 software development security practices. We applied the results of the mapping study and practices review, to build the SP-EF framework and SOTM, which are presented in Chapter 4.
2. *Case Study* [66] (RQ2): We conducted a case study of security practice use in a typical software development project at IBM. We collected SP-EF measurements from three perspectives: qualitative researcher observations, a survey of the team members, and text mining of the team’s development history.
3. *Survey* [67] (RQ2): We surveyed 11 security-focused open source projects to collect SP-EF security practice adherence data and analyzed the practice adherence metrics.
4. *Model Evaluation* (RQ3): We evaluate the SOTM and our hypotheses quantitatively, applying Structural Equation Modeling (SEM) to 697 projects reported on in two public datasets, OpenHub, and the National Vulnerability Database (NVD).
5. *SP-EF Longitudinal Case Studies* (RQ2, RQ3): We apply SP-EF to software development project history from Bitcoin, IBM wasCloud, phpMyAdmin, and OpenSSL to observe practice adoption and outcome measures over time.

1.4 Contributions

This dissertation makes the following contributions:

- The SOTM, a proposed model of the constructs affecting software development security outcomes;

- SP-EF, a proposed set of measures for assessing security in the software development process, and
- Empirical evaluation of the proposed model and metrics using open source and industrial datasets.

Chapter 2

Background and Related Work

To provide grounding for the topic of security metrics in software development, our mapping study, and our classification scheme, this section presents a glossary of metric-related terms, and literature on software metrics generally, and specifically on software security metrics.

2.1 Definitions

Attribute: a property associated with a set of real or abstract things that is some characteristic of interest. [31]

Attack: An intentional act by which an entity attempts to evade security services and violate the security policy of a system; A method or technique used in an assault [87].

Fault: an incorrect step, process, or data definition in a computer program. [31]

Indicator: Any observable characteristic that correlates with a desired security property [81].

Measure: A way to ascertain or appraise value by comparing it to a norm; To apply a metric [32].

Measurement: The process by which numbers or symbols are assigned to attributes of subjects in the real world in such a way as to describe them according to clearly defined rules [20].

Metric: A quantitative measure of the degree to which a system, component, or process possesses a given attribute; the defined measurement method and the measurement scale [31].

Risk: The combination of the probability of an event and its consequence ¹.

Security metric: A metric which measures a security property [81].

¹ISO/IEC 16086 (IEEE Std 16085-2006) - Systems and Software Engineering - Life Cycle Processes - Risk Management

Security property: A property related to security concerns, for example confidentiality, integrity, availability, authentication, authorization, non-repudiation [52].

Software security: We adopt McGraw’s notion of “engineering software so that it continues to function correctly under malicious attack” [52]. For our notion of malicious attack, we also reference the IEEE definition of software security: ”Protection of information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them.” ¹⁰

Vulnerability: A fault or weakness in a system’s design, implementation, or operation and management that could be exploited to violate the system’s security policy [87].

2.2 Measurement Frameworks

Measurement frameworks offer a foundation for aggregating study results by standardizing the collection of empirical evidence, enabling comparison of results across projects. Previous researchers [36, 98] have suggested four reasons for measurement frameworks:

- To allow researchers to provide a context within which specific questions can be investigated;
- To help researchers understand and resolve contradictory results observed in empirical studies;
- To provide researchers a standard framework to assist in data collection and reporting of empirical studies in such a manner that they can be classified, understood, and replicated and to help industrial adoption of research results;
- To provide researchers a framework for categorizing empirical studies into a body of knowledge.

Williams et al. [98] defined a measurement framework for evaluating the use of Extreme Programming (XP). The Extreme Programming Evaluation Framework (XP-EF) contains context factors to capture internal project-related variables; adherence metrics to capture XP practice use; and outcome measures to capture external project results (e.g. quality). Modeled after XP-EF, we have developed SP-EF, a measurement framework for software development security practice use. We defined a similarly structured set of measures for recording context factors, practice adherence metrics, and outcome measures, related to the use of security practices in software development. Rudolph and Schwartz [81] conducted a systematic literature review,

and Morrison et al. [65] produced a systematic mapping study on security metrics. We chose the context factors and outcome measures aligned with the findings of these studies.

The Common Vulnerability Scoring System [54] (CVSS) is a framework for communicating the characteristics of vulnerabilities in information technology (IT). Our focus is on the software development process and product rather than the individual vulnerability, however we adopt the Confidentiality Requirement, Integrity Requirement, and Availability Requirement elements of CVSS as context factors.

2.3 Structural Equation Modeling

Structural Equation Modeling (SEM) is a family of statistical techniques for testing theories by specifying models that represent relationships among theoretical constructs, as measured by appropriate observed variables [37]². While most traditional statistical methods emphasize the modeling of individual observations, SEM emphasizes the covariances of the observed variables [27]. In SEM, a model is posited that specifies the relationships among variables, resulting in systems of equations where the relationships between all variables are linear (or transformably linear). Hildreth [27], gives a clear presentation of the linear algebra involved in specifying and solving SEM models. In the equations shown in Table 2.1, the relationships between latent variables internal (endogenous, η) and external (exogenous, ξ) to the model are represented by matrices of the endogenous (β) and exogenous (Γ) structural parameters containing the coefficients for the expected (modeled) relationships. Based on these equations, the population covariance matrix of the observed variables can be represented as a function of the model parameters. The population covariance matrix is derived from datasets representing a sample drawn from the population, and estimation algorithms are applied to the model and dataset to solve for the model parameter estimates.

We rely, in particular, on two features of SEM to support our investigation: the distinction between observed variables and latent variables, and the distinction between structural and measurement models.

Not every quantity we wish to measure can be measured directly. For example, in psychological studies of ‘general intelligence’, g , researchers examine correlations between multiple observed test scores and the notion, or ‘construct’ of general intelligence. Latent variables are a tool for analyzing combinations of observed variables that correspond to hypothetical constructs, which are presumed to reflect a continuum that is not directly observable [37].

²Suhr provides a concise online tutorial: <http://www.lexjansen.com/wuss/2006/tutorials/TUT-Suhr.pdf>

Table 2.1 SEM Notation, adapted from Hildreth [27]

<i>Structural Model Equation</i>		
$\eta = \beta\eta + \Gamma\xi + \zeta$		
<i>Assumptions</i>		
$E(\eta) = 0$		
$E(\xi) = 0$		
$E(\zeta) = 0$		
$E(\xi\xi') = 0$		
$(I - \beta)$ is nonsingular		
Symbol	Dimension	Definition
<i>Variables</i>		
η	$m * 1$	endogenous latent variables
ξ	$n * 1$	exogenous latent variables
ζ	$m * 1$	latent errors in equations
<i>Coefficients</i>		
β	$m * m$	coefficient matrix for endogenous latent variables
Γ	$m * n$	coefficient matrix for exogenous latent variables
ϕ	$n * n$	covariance matrix of ξ
ψ	$m * m$	covariance matrix of ζ

Researchers commonly investigate the effects of unobservables like intelligence by statistically relating covariation between observed variables to latent variables [9]. We adopt latent variables as a suitable tool for relating observed data to the constructs we study, for example Usage Risk.

The distinction between structural and measurement models parallels the distinction between latent variables and observed variables. Structural models are representations of the latent variables and the relationships between them. Measurement models are representations of observed variables and the relationships between them. A measurement model may be linked to a structural model to indicate relationships between observed and latent variables. Repeated studies with a single structural-measurement model combination contribute evidence for accepting or rejecting the combined models. A single structural model may be associated, serially, with a variety of measurement models, reflecting a variety of approaches to measuring a theoretical construct. Observations of similar trends using different data sets and measurement approaches lends strength to the underlying theory [7, 101].

A number of software packages, including `sem`³, `MPlus`⁴, and `lavaan` [80] (LAtent VArIable ANalysis), provide tools for specifying and estimating SEM models. SEM studies are organized around six steps (which may be iterated over, as called for by the needs of the study):

- *Model specification* Express the hypothesized relationships between observed variables and latent variables, typically in the form of a graphical model. Each edge in the graph represents a parameter to be estimated, indicating the strength of the relationship between the nodes connected by the edge.
- *Identification* Check the specified model against statistical theory for whether all of the models parameters can be estimated given the model’s structure. Identification is analogous to checking whether a set of equations is solvable given their structure and the data at hand. If the original model cannot be identified, it must be revised in light of both statistical theory and the theory the researcher is expressing in the model.
- *Data selection* Choose data for each of the models observed variables. Kline [37] states that SEM is a large-sample technique, reporting median sample size in the literature of 200 cases, starting with a minimum of 10 observations per parameter to be estimated. Factors that drive the need for a large sample include the size and complexity of the model to be estimated, non-normally distributed data, categorical data, and imprecisely measured data.

³<https://cran.r-project.org/web/packages/sem/index.html>

⁴<https://www.statmodel.com/>

- *Data collection* Obtain data from the study’s sources, and, if necessary, clean and transform the data.
- *Estimation* Check the observed data and the model for fit. If appropriate fit is achieved, the parameter estimates can be interpreted for implications of the theorized relationships and the observed data. If appropriate fit is not achieved, the list of model changes developed during specification should be considered in re-specifying the model.
- *Reporting results* Report the model, parameter estimates, fit measures, and any changes made for re-specification.

Capra et al. [12] applied SEM to examine the relationship between design quality, development effort and governance in open source projects. Wallace and Sheetz [95] applied SEM to study development team adoption of software metrics. Gopal et al. [25] applied SEM to study how institutional factors affected metrics program adoption.

2.4 Software Development Security Metrics

We require a set of metrics to support measurement of security practice adherence, security-related context factors and security outcomes. Software engineering authors have used multiple definitions for the words “metric” and “measure”. For the purposes of this dissertation, we are liberal in the definitions we accept from the literature. A good metric should be conceptually specific, quantitatively measurable, practically attainable, consistently measured without subjective criteria, and time-dependent [76]. However, even when metrics appear to be useful, difficulties arise attempting to validate metrics and determine their overall usefulness and practicality [83]. In addition, metrics, including security metrics, are not valuable if the results of applying them cannot be understood effectively [71].

What is a *security metric*? Jansen [32] quotes and discusses three variant definitions from the literature, highlighting a diversity of usage and a reliance on human judgment rather than more repeatable measures. For our purposes, we define a security metric to be a metric whose attribute is a measure of a security property, or some indication of a violation of a security property. We limit our focus to the artifacts of software development; software, the source code from which software is produced, and the related test suites, documentation, and other artifacts used and created during the software development process.

Verendel [93] presents a survey focused on measuring operational security, addressing the ability to “function correctly under malicious attack.” Our mapping study additionally considers

the engineering of secure software, seeking measurements of the process, tools, people, and the software produced. Rudolph and Schwarz [81] surveyed scholarly articles on “security indicators”, where an indicator is defined as “an observable characteristic that correlates with a desired security property.” In addition to what Rudolph and Schwarz studied, we seek to characterize the subjects being measured. Meneely, Smith, and Williams [58] review metric validation, and suggest a scheme for choosing validation criteria. We only consider validation in terms of the high-level approach chosen by the researchers, e.g., whether the metric(s) were evaluated through user study, theoretical analysis, or researcher opinion.

Savola’s [82] security metrics taxonomy, characterizes security metric properties and applications. At a high level, Savola provides three categories of metrics:

- Organizational metrics describe attributes of organizational programs and processes.
- Technical metrics describe software artifacts, e.g., requirements, specifications, designs, code.
- Operational metrics describe running systems and their environments.

To establish a set of security metrics possessing the properties described above and suitable for SP-EF, we conduct a mapping study, described in Section 3.

2.5 Software Development Security Practices

We draw our base of software development security practices from the BSIMM, SDL, OWASP, and SSAP listed in the introduction. We identified these sources by searching for ‘software development security practices’ at IEEE Xplore ⁵, and identifying lists of security practices for software development that have been applied in industrial software development settings. While other lists of software development security practices exist, particularly those defined by government standards bodies, each of these four sources has been derived from the experiences of software development teams, and so each of the practices identified have an existing empirical basis. We now present each source in more detail.

2.5.1 The Building Security In Maturity Model

The BSIMM [51] is a survey of software security initiatives. BSIMM’s goal is to allow comparison within and between organizations, and over time, along the security practice dimensions identified in the BSIMM. BSIMM organizes the software security development life cycle (SSDL) into

⁵<http://ieeexplore.ieee.org/>

four categories; Governance, Intelligence, SSDL Touchpoints, and Deployment. BSIMM defines 112 activities (practices in our terminology) in which the participating organizations engage. BSIMM does not specify criteria for how to select or when to apply the security practices.

2.5.2 The Microsoft Security Development Lifecycle

Bill Gates’ Trustworthy Computing Memo of 2002 ⁶ spurred investment within Microsoft in security training, tools, and process, eventually described as the Security Development Lifecycle (SDL). The goal of the SDL is to minimize security-related vulnerabilities in design, code, and documentation and to detect and eliminate vulnerabilities as early as possible in the development lifecycle. Howard and Lipner [28] presented an overview of the SDL, and a detailed description of the SDL is available online ⁷. Microsoft has also developed a “Simplified Implementation of the SDL” (SISDL) document ⁸ illustrating the SDL’s core concepts and required security activities. In the SDL and SISDL, the development lifecycle is defined to have five phases: Training/Policy/Organizational Capabilities, Requirements and Design, Implementation, Verification, and Release and Response. The SDL defines “Security Advisor” and “Team Champion” roles in addition to the standard Microsoft roles of developers, testers, and managers.

2.5.3 SAFECODE

The SAFECODE initiative Fundamental Practices for Secure Software Development [89] (SAFECode) lists “foundational” secure development practices that have been reported as being effective in improving software security in real-world implementations by members of the SAFECODE initiative. SAFECODE identifies two roles: Developers and Testers. SAFECODE lists 17 security practices that should be applied during three phases of software development: Design, Programming, and Testing. For each practice, SAFECODE includes a section on verification listing tools and techniques for verifying that the practice has been correctly applied.

2.5.4 OWASP CLASP

The Open Web Application Security Project (OWASP) Software Security Assurance Process (Comprehensive, Lightweight Application Security Process) CLASP ⁹ is a prescriptive set of

⁶<http://news.microsoft.com/2012/01/11/memo-from-bill-gates/>

⁷<http://www.microsoft.com/en-us/download/details.aspx?id=29884>

⁸<https://www.microsoft.com/en-us/download/details.aspx?id=12379>

⁹https://www.owasp.org/index.php/Category:OWASP_CLASP_Project

software development security process recommendations. CLASP lists 24 activities (practices) to be applied across six phases; Policy, Specification, Analysis and Design, Implementation, Test and Operation, and Maintenance.

2.5.5 Other Security Practice Lists and Surveys

McGraw [52] defined a set of seven Touchpoints for addressing security concerns during the software development process: Code Review, Architectural Risk Analysis, Penetration Testing, Risk-Based Security Testing, Abuse Cases, Security Requirements, Security Operations, and External Analysis.

De Win et al. [100] compared CLASP, the SDL, and McGraw’s Touchpoints frameworks for software development security practices and evaluated the framework strengths and weaknesses during Requirements and Design phases on a case study. They concluded that the SDL’s strength was in architectural threat analysis, Touchpoints’ strength was analysis-level risk modeling, and CLASP’s strength was in architectural design. We examine the frameworks to identify and measure common practices across the full life cycle, to augment the identified practices with measures of their usage, and to empirically validate our measurements with software development teams.

Epstein [18] conducted interviews in software development organizations to identify software development security “best practices”, finding that larger organizations tended to apply more security practices than smaller organizations, and that nearly every organization interviewed practiced penetration testing and developer training in security. Epstein [18] focused on interviewing commercial producers of “shrink-wrapped” software, while we develop a survey for both commercial and open source projects.

Baca and Carlsson [6] interviewed developers in a commercial organization about integrating SDL, McGraw Touchpoints, and Common Criteria practices into an agile development process. They found that the developers believed the three security practice frameworks were too expensive or of not enough value to implement. At the level of individual practices, they believed that developing Security Requirements and Abuse Cases could be applicable to their process, but that Static Analysis did not produce enough value for the effort required to identify false positives. We build a survey of practice adherence rather than an interview protocol.

Futcher and von Solms [22] reviewed software development security practice recommendations from standards bodies, producing a summarized list of recommendations. In terms of software process management, they identified the need to incorporate security during each phase rather than adding security in later, the need to define security-specific roles, and the need for

security training. In terms of software process elements, they identified security requirements, threat modeling, coding standards, code reviews, and security testing tools. They perform no empirical evaluation on the recommendations. We identified a superset of their practices in our study, and develop a survey to capture team data on practice adherence.

Oyetoyan [72] surveyed security practice use in agile teams, asking five questions about a list of practices identified by Ayalew et al. [5] and collected data from two industrial projects. They found that developer skill correlated with practice application, that some practices were infrequently used, and that developers sought further training in security practices. We survey software development teams in general, rather than only agile teams, extract a set of security practices from literature review, and augment the practices with adherence metrics.

Wheeler ¹⁰ developed a checklist of recommended security practices for open source projects through literature review, measuring adherence by number of items checked off for each project. The criteria are embedded in an application, “BadgeApp”, to award open source projects for compliance with the listed practices. We surveyed BadgeApp developers as a check on our practice list.

2.6 Technology Adoption

Researchers have developed models for how technology users choose new technologies, referred to as “technology adoption” models. Davis [16] developed and validated scales for perceived usefulness and perceived ease of use, finding correlation for both variables with user behavior. Davis defined perceived usefulness as “the degree to which a person believes that using a particular system would enhance his or her job performance”, and perceived ease of use as “the degree to which a person believes that using a particular system would be free from effort”. Davis’ [15] Technology Acceptance Model (TAM) posits perceived usefulness and perceived ease of use as constructs central to a user’s intention to use a given technology, and TAM has been applied and reviewed extensively, see Legris [44] and Venkatesh [92] for examples. Venkatesh [92] examined TAM and seven related models and extensions to TAM, and proposed the Unified Theory of Acceptance and Use of Technology (UTAUT) model. UTAUT retains the TAM constructs, while adding Social Influence and Facilitating Conditions as primary constructs driving user intention to use a given technology. Our security practice adherence measures reflect the TAM and UTAUT line of work.

¹⁰<https://github.com/linuxfoundation/cii-best-practices-badge/blob/master/doc/criteria.md>

2.7 Practice Adherence

Different projects are unlikely to use the same set of security practices, or to use a given security practice in exactly the same way. Adherence metrics are a means of characterizing the degree to which a practice is used on a project. Williams et al. [97] proposed a set of objective and subjective adherence metrics for assessing the degree to which a team held to Extreme Programming (XP) practices. XP-EF's objective metrics are based on counts of new and changed classes and test cases, and on frequencies of planning, test runs, inspections, and pair programming. XP-EF's subjective metrics are based on survey responses to the Shodan Adherence survey [39], which contains multiple-choice questions on the degree of individual use for each of the XP practices. Layman [43, 42] reports on two case studies of XP-EF application. We emulate the organization and intent of the XP-EF adherence metrics, while adapting the content to focus on security practice adherence rather than XP, and adapting the metrics to support data collection for technology acceptance theory.

Wood et al. [102] studied XP adherence and its relationship to teamwork in 40 student/professional XP teams, using an adapted version of the Shodan survey, finding correlation between XP adherence dimensions and team performance. However, adherence to the Foundations practices (automated unit tests, customer acceptance tests, test-first design, pair programming, and refactoring.) was negatively correlated with performance in some situations, suggesting that practices have costs as well as benefits. Wood's constructs and results inform our modeling techniques.

Chapter 3

Security Metrics and Practices Mapping Study

3.1 Introduction

In this chapter ¹ we review software development security metrics and security practices to form a basis for answering ‘*RQ1* What measurements (context factors, security practices, adherence measures, and outcome measures) affect software development security outcomes?’.

We conducted separate reviews of software development security metrics and security practices, as described below.

3.1.1 Security Metrics Systematic Literature Review

Providing useful metrics for the security of a software system is a difficult undertaking because: 1) we lack effective models and measurements of software risk; 2) security must be considered from the earliest stages of software development; 3) the people building and using the software must be considered in estimation of security risk and assurance [21]. Many elements of the software development life cycle contribute to software security. Pfleeger and Cunningham [75] consider dimensions ranging from the specification of systems to protocol verification to the psychology of software designers, users and attackers. The range of dimensions suggests that a wide range of metrics is needed to properly represent security for assessment and prediction.

As a means of identifying the security properties measured, we conducted a systematic mapping study of the metrics that have been applied to measuring the security in the software

¹Portions of this chapter appear in Morrison et al. [65]

development life cycle. According to Budgen [10], systematic mapping studies are “intended to ‘map out’ the research that has been undertaken rather than to answer a detailed research question.” We include metrics measuring the software, and the artifacts, processes and people involved in the software development life cycle, as well as metrics measuring aspects of security (e.g. measures of Confidentiality) or its absence (e.g. counts of vulnerabilities). We term these metrics ‘software development security metrics’.

To assess software development security metrics and their evaluation and use, we subdivide RQ1 into the following research questions to map out the state of the field:

RQ1.1: What software life cycle security metrics have been proposed in the literature?

RQ1.2: What are the people, artifacts, and process elements being measured by software life cycle security metrics?

RQ1.3: What validation approaches are used for software life cycle security metrics?

RQ1.4: During what phases of the software development life cycle are software life cycle security metrics measured?

Our initial search yielded a set of 4,818 papers. We narrowed the set to 71 papers that propose, evaluate and/or report on 324 security metrics for the software life cycle.

3.1.2 Security Practices Review

In addition to the security metrics mapping study, we reviewed four lists of software development security practices, in Section 2.5, to assess their commonalities and differences. We formulated a research question to this end:

- **RQ1.5** What software development security practices are used by software development teams?

Our contributions include:

- A systematic map of the metrics used to evaluate the security properties of software and its development and use.
- An online catalog of software development security metrics.
- A list of software development security practices.
- A set of practice adherence measures.

The remainder of this chapter is organized as follows: Section 3.2 describes the methodology we followed in executing the mapping study. Section 3.3 provides our summarization of the data collected. Section 3.4 presents our discussion of the results, and Section 3.5 reports on Limitations. Section 3.6 concludes the security metrics mapping study. Section 3.7.1 presents our review of the BSIMM, SDL, SAFECODE, and OWASP security practice recommendation lists.

3.2 Methodology

We subdivide how we approach the mapping study into four components: our search strategy for identifying papers, our selection criteria for including papers, our classification scheme for collecting data on each metric, and our procedure for extracting metric information from each paper.

3.2.1 Search Strategy

In this section, we describe the process used to conduct our systematic mapping study.

3.2.1.1 Databases

We based our selection of online databases on the most common databases used in Software Engineering Systematic Literature Reviews (SLRs), and in Systematic Mapping Studies (SMSs), and on sources used in previous software security metric literature reviews [81], [71]. The data sources in this study include online databases, conference proceedings, and academic journals. The list is as follows: ACM Digital Library, IEEE Xplore, and Elsevier.

3.2.1.2 Search Terms and Strategy

We developed a list of search terms with which to evaluate titles, shown in Table 3.1. For each term associated with a research question, we identified synonyms in the titles and abstracts of previous surveys [81], [93] of security metrics. The synonyms are shown in Table 3.2. Two researchers developed terms and synonyms independently, and then combined lists, retaining terms and synonyms agreed on as most relevant.

After searching using various combinations of the proposed keywords and evaluating their performance using the set of papers in our Quasi-Gold-Standard (QGS, Section 3.2.1.5), we selected our basic search phrase: Our base search phrase is:

(software OR computer OR information)

Table 3.1 Research Question Keywords

Research Question	Keyword
RQ1.1	“software”, “security”, “metric”
RQ1.2	RQ1.1 terms
RQ1.3	RQ1.1 terms + “validate”, “evaluate”
RQ1.4	RQ1.1 terms + “phase”, “lifecycle”, “process”

Table 3.2 Keyword Synonyms

Keyword	Synonym(s)
software	“application”
security	“vulnerability”
metric	“measurement ”, “measure ”, “indicator ”, “attribute ”, “property ”
validate	“validation ”, “evaluate ”, “evaluating ”, “evaluation ”, “quantify ”, “quantifying ”, “quantification ”, “quantified ”, “quantitative ”, “assess ”, “assessment ”, “measure ”, “measurement ”, “appraisal ”, “analyze ”, “measuring ”, “analysis ”, “assurance ”, “scoring ”

AND (security OR vulnerability OR assurance)

AND (metric OR indicator OR quant*)

We created database search phrases using the base search phrase and syntax appropriate to each database and collected papers based on those search phrases.

3.2.1.3 Search Process Evaluation

We followed Zhang [104] in evaluating the quality of our search results. An ideal search process would return all relevant papers (sensitivity = 1.00), and only relevant papers (precision = 1.0). Such a set of papers would be a “gold standard.” In practice, we do not know the set of relevant papers in advance, so we must estimate, using what Zhang terms a “quasi-gold standard” (QGS) which is a set of relevant papers in the literature, chosen prior to the search process. The QGS is used as a measure of how well each search string locates relevant papers. Zhang [104] defines sensitivity as the ratio of the number of retrieved papers to the total number of relevant studies. Quasi-sensitivity (QS) is the number of papers returned by a search to the number of returned papers that are present in the QGS. QS estimates how well the search string locates relevant papers within the searched corpus. By measuring the performance of the search in returning members of the QGS from a given search engine, compared with the number of QGS papers in the search engine, an estimate can be made of search performance. For example, if there are 10 of the QGS papers in the ACM library, and the search string returns 8 of them, QS would be

0.8.

3.2.1.4 Selection Criteria

We developed a list of criteria to assess whether the papers found during the search process met our objectives.

1. Inclusion Criteria:

- Related to measuring software security in software development artifacts and the software development process.
- Measurements and/or metrics are main subject
- Refereed publication
- Published since 2000

2. Exclusion Criteria

- Sources measuring security in non-software systems
- Sources related to identity, anonymity, privacy
- Sources related to forgery and/or biometrics
- Sources related to network security
- Sources related to encryption
- Sources limited to database security
- Books, dissertations

3. Study Quality Assessment: We developed a Quality Assessment Checklist for whether to include each paper, as follows:

- Is a primary or secondary goal of the paper to describe, define, or evaluate a metric or measurement of software security?
- Does the paper align with our inclusion criteria?
- Is the paper peer-reviewed?

4. Scoring: We also established a scoring procedure for resolving differences between raters when disagreement was found.

- Question Scoring Scale: No: 0, Partial: 0.5, Yes: 1, Two raters.

- Complete agreement
 - (a) Yes from both raters: paper is selected
 - (b) No from both raters: paper is rejected
- Partial agreement combinations between 0 and 2. Raters discuss, find agreement, or agree to disagree.
 - (a) Agreement processed according to the rules for complete agreement.

In the case of unresolved disagreement, papers are selected for evaluation in the subsequent step in the protocol.

3.2.1.5 Search Results

The second author and I identified a set of 39 software development security metrics papers, developed by reviewing the papers collected in the previous security metrics literature reviews [81], [93]. Each author read the titles and abstracts of the collected papers independently, applied the study quality assessment criteria, and made a list of candidates. The authors then discussed each list, applying the following selection procedure:

1. If both authors agreed the paper described a software development security metric, it was included in the final QGS list.
2. If both authors agreed the paper did not describe a software development security metric, it was excluded from the final QGS list.
3. Disagreement was discussed. If agreement could be reached, the appropriate action listed above was taken. If agreement could not be reached, the paper was included in the final list.

After our process, the QGS consisted of 17 papers. The results of each database search were compared with the QGS set of papers. The quasi-sensitivity for each database search is reported in Table 3.3.

We conducted three passes over the collected papers, applying the Study Quality Checklist and scoring procedure to each paper’s title, abstract, and a 10-minute ‘quick read’ of each paper. After reviewing the titles and coming to agreement, the raters (first two authors) identified 4815 papers to continue investigating. After reviewing the abstracts, the raters identified 107 papers to continue investigating. After giving each paper a ‘quick read’ and coming to agreement, the raters identified 71 papers for further investigation. We assigned a reference code to each of

Table 3.3 Search Results

Database	Initial Search	QS
ACM Digital Library	2302	0.77
IEEE Xplore	2000	0.80
Elsevier	1280	0.63

the 71 papers, P1-P71. We use these codes throughout the remainder of the paper to refer to individual papers. See Table A.4 for the reference codes and bibliographic references of the included papers.

3.2.1.6 Metric Classification Scheme

To support answering our research questions, we developed a set of data elements to be collected for each metric.

We begin with the IEEE definition of metric, “A quantitative **measure** of the degree to which a system, component, or process possesses a given **attribute**; the defined measurement **method** and the measurement **scale**” [31], defining the following data elements to describe each metric:

- Metric name: the “given name” of the metric defined or used in the paper. (RQ1.1)
- Measure: A description of how the value of the metric is determined, in terms of the metric’s Method and Scale (RQ1.2, RQ1.3).
 - Method: Classification of how the metric value is measured, where “Quantitative” indicates objective, systematic, empirical measurements and “Qualitative” indicates subjective measurements based upon observation (RQ1.3).
 - Scale: denotes type of measurement scale (e.g., Nominal, Ordinal, Interval, Ratio), or specific instance of measurement scale (e.g., Count, Probability, Duration, Rate, Currency, Percentage) (RQ1.2).
- Subject Type: Classification of whether the metric measures a ‘System’, ‘Component’, or ‘Process’ (RQ1.2). Target: class of subject measured by the metric (R&S).
 - Product: Refers to security-related elements of a target, e.g., software products or parts of them source code, components, systems, etc.

- Process: Refers to security-related elements of a process, e.g., a development process, or a maintenance process.
- Resources: Refers to security-related attributes of resources used by a product or process.
- Subject: Description of the entity, ‘an object or event in the real world’ (Fenton [20]), being measured. Extracted from papers, or synthesized and named based on assessment of extractors. For example, ‘Source Code’, ‘Component’, and ‘Vulnerability’. The complete list is provided in Results, Section 3.3.2. (RQ1, RQ2)
- Attribute: Description of ‘a feature or property’ (Fenton [20]) of the subject being measured. Extracted from papers, or synthesized and named based on assessment of extractors. For example, ‘Lines of Code’ or ‘Classified’ (RQ1.2).
- Validation Approach: To characterize how metrics are validated, we record how the metrics are evaluated by the researchers in the paper. Values: Opinion, Theoretical, Academic user study, Industry user study, Reports from production (RQ1.3).
- Phase: We record the phase in the software development lifecycle with which the metric is associated: Requirements, Design, Implementation, Testing, Operations (RQ1.4).

For demographic purposes, we assigned unique paper and metric numbers to identify each metric and its usage across papers. For audit purposes, we tracked the name of the extractor and auditor for each row, as well as the extraction and audit dates.

3.2.1.7 Data Extraction

The metric extractor (first or second author) reads a paper, identifies each metric defined or used in the paper, and collects the data for each element in the metric classification scheme. The first author applied the extraction procedure described in the classification guide to every paper in the final set of 71 papers, with spot checks by the second and third authors.

3.2.1.8 Metric Categorization

To discover relationships between similar metrics with varying names in different papers, we applied card sorting [30] to categorize the metrics. Card sorting is a technique for identifying commonalities between collections of data items by using multiple raters to compare and group related data items. Over the course of four sessions, the first, second, and third authors discussed

each metric and its definition, and place it in a proposed category and subcategory. When a metric did not fit into existing categories, we created a category or subcategory. Over the course of categorizing the metrics, we moved metrics between categories, and merged or removed categories to capture our shared understanding of the metric definitions. After we assigned metric to a category, we took another pass, reviewed each metric and category definition and revised our choices.

3.3 Results

In this section, we present our results in the form of summaries of the data extracted from the selected papers. Based on the data extraction procedure, we tabulated the metrics, subjects, scales, evaluation means, and uses. The tabulated data provides an overview of ‘evidence clusters’ and ‘evidence deserts’ for software security in the software development lifecycle. The full data set is available online ².

3.3.1 RQ1.1: What software lifecycle security metrics have been proposed in the literature?

In total, we identified 324 unique metrics in the 71 selected papers. The list of metrics is available online ³. We present each metric category and subcategory we identified (as described in Section 3.2.1.8), ordered by the number of metrics in each category/subcategory. We do not present each metric, but we give representative examples of metrics from each category. Table 3.4 contains our final list of eight categories with associated subcategories, with metric counts for each category and subcategory. The following sub-sections will discuss each of the eight categories.

3.3.1.1 Security Properties Metrics (86)

We identified metrics as part of the Security Properties Metrics category when the metric is primarily a measure of a security property. We identified five subcategories of Security Property metrics: Access Control Metrics, Audit Metrics, Availability Metrics, Confidentiality and Privacy Metrics, and Integrity Metrics.

²https://github.com/pjmorris/SMS_SecurityMetrics/blob/master/Data%20Extraction_Take3.xlsx

³https://docs.google.com/spreadsheets/d/1cBfmWY_Ab0e3TuN5z0wtiNp4YJh7MDSWxDExEt7qnG0/edit#gid=1939289276

Table 3.4 Security Metric Categories and Subcategories

Metric Category	Subcategory	Subcat Total	Cat Total
Security Properties			86
	Access Control	42	
	Confidentiality and Privacy	19	
	Audit	9	
	Availability	6	
	Integrity	10	
Implementation			84
	Source Code	44	
	Security Source-Code	30	
	Version Control	8	
	Data Flow	2	
Project Management			41
	Process	17	
	Design	17	
	Defect	7	
Incident			36
	Vulnerability	20	
	CVSS	16	
Resource			34
	Cost	9	
	People	23	
	Time	2	
Defensibility			19
	Attack Surface	9	
	Attackability	10	
Aggregate			18
	Aggregate	18	
Security Requirement			16
	Requirement	4	
	Usecase-Misusecase	12	
Grand Total			324

Access Control Metrics (42) Papers P2 and P9 present lists of questions to verify the presence of desirable access control mechanisms in the system being measured, e.g. ‘Authentication non-repudiation’, ‘Authorization’, ‘User authority’, and ‘Intrusion detection’. Paper P33 defines categorical metrics for detailing how authentication and authorization are implemented in the system being described, e.g. ‘User authentication scheme’ and ‘User identification scheme’ (e.g. token, password). Paper P38 expands on the idea of ‘User identification scheme’ to define 23 metrics for describing how passwords are implemented in a system, e.g. ‘Alphabet Size’, ‘Maximum Life Time’, and ‘Users Training’.

At a more abstract level of design, P17 defines ‘Compartmentalization’ (M73), ‘the number of independent components that do not trust each other (performs authentication and authorization for requests/calls coming from other system components) that the system is based on to deliver its function.’

Operational metrics for access control focus on tracking successful and unsuccessful access attempts on system resources. ‘Monitoring system use’ (M170, P4) is a count of unauthorized access attempts on system resources, and ‘Login Attempt Count’ (M162, P33) is a count of login attempts.

Audit Metric (9) Papers P2 and P33 present lists of questions to verify the presence and quality of audit mechanisms. ‘Auditing’ (M33,P2) is a question verifying the presence of techniques or technology for recording ‘who did what, when and where?’ ‘Audit trail comprehensiveness’ (M34, P33) is a question verifying that all failed authentication and authorization attempts are logged. ‘Log File Protection Scheme’ (M161, P33) is a check on the presence of a scheme for maintaining log file integrity. Operational Audit Metrics include ‘Ratio of log interactions to system interactions’ (M209,P33) and ‘Audit logging’ (M33, P4) the ratio of log files that are monitored to the total number of log files.

Availability Metrics (6) Papers P2 and P33 present lists of questions to verify whether availability concerns are addressed in the system being measured. ‘Availability’ (M100, P2) is a question verifying that a denial of service mitigation scheme is in place. ‘Critical Service Interaction’ (M99,P33) is a count of the services depended on by the system being measured.

Confidentiality and Privacy Metrics (19) Papers P2, P23, and P33 present lists of questions to verify the presence and quality of mechanisms supporting confidentiality. ‘Confidentiality security attribute’ (M83, P23) includes a set of requirements for maintaining confidentiality ‘in terms of required behavior during execution expressed in terms of data and transformation of data by programs.’ ‘Side-channel Vulnerability Factor’ (M249, P59) ‘measures information leakage through a side-channel by examining the correlation between a victims execution and an attackers observations.’

Integrity Metrics (10) Papers P2 and P33 present lists of questions to verify whether integrity concerns are addressed in the system being measured, covering, for example, ‘Data integrity’ (M148, P2), and ‘Validation checks per input’ (M143, P33). The Common Vulnerability Scoring System [55] (CVSS) ‘Integrity Impact’ metric (M146) is a categorical measure (None, Partial, Complete) of the impact of a vulnerability on the integrity of the affected software. ‘Integrity’ (M145, P30) is the ratio of ‘risky’ classes to total classes, where ‘risky’ is defined by a set of attributes of the measured class, e.g. ‘references user data’, ‘references password’.

3.3.1.2 Implementation Metrics (84)

We identified four subcategories of Development/Implementation Metrics: Data Flow Metrics, Source Code Metrics, Security Source Code Metrics, and Version Control Metrics. We now discuss each of these subcategories.

Data Flow Metrics (2) Two papers, P45 and P46, define data flow metrics for identifying vulnerable code.

Source Code Metrics (44) We found 84 references to traditional source code metrics, e.g. Source Lines of Code (SLOC), Churn, McCabe’s Cyclomatic Complexity, and Coupling, in 21 papers: P1, P5, P9, P18, P19, P25, P28, P30, P33, P36, P44, P46, P47, P53, P55, P58, P60, P61, P68, P69, P71. We include object-oriented metrics in this category. While we focus on metrics of security properties in this paper, we make four observations on how source code metrics are applied in the context of security. First, traditional source code metrics have been used in varying combinations to predict vulnerabilities, so far with limited success. Second, traditional source code metrics are used to normalize security concerns to code size. For example, ‘Relative Churn’ (M53, P71) normalizes churn to source file size. Third, traditional source code metrics are often ‘tweaked’, or ‘adapted’, e.g. 30-Day Churn (M53, P71), Min Churn, Max Churn, Mean Churn, to broaden studies of the correlations between the base metrics and the dependent variables under study. Fourth, traditional source code metrics are sometimes augmented with security information to yield new metrics, for example, the ‘Classified’, and ‘Critical’ labels on attributes, methods, etc in P1.

Security Source Code Metric (30) Paper P1 defines the largest set of security source code metrics among the papers surveyed. In the framework designed by the researchers, attributes are marked as ‘classified’, as determined by the developer, and classes are marked as ‘critical’ if they contain classified attributes. The researchers build up a system of metrics for classes and methods that read and write the classified attributes and interact with critical classes, and then define a set of metrics representing various security properties in terms of the

lower level definitions. For example, ‘Classified Attributes Inheritance’ (M55, P1) indicates that a class inherits classified attributes from another class, and ‘Critical Class Extensibility’ (M94, P1) indicates that a critical class is not final. Similarly, ‘Variable Vulnerability’ (M291, P5) is based on marking variables as security-relevant and evaluating their relationships.

Version Control Metric (8) The authors of Paper P68 uses metrics like ‘Commit Count’ (M71), ‘Star Count’ (M260), and ‘Fork Count’ (M130) for subject selection of repositories for vulnerability analysis.

3.3.1.3 Project Management Metrics (41)

We identified two subcategories of Project Management Metrics: Design Metrics, and Process Metrics. We now discuss each of these subcategories.

Design Metric (17) Paper P9 presents a series of checklist metrics for kernel management, component interfaces, and network planning. Paper P19 presents a series of metrics for measuring each stage of the software development lifecycle, including measures of design effort, security design decisions, and the ratio of security design decisions to total design decisions.

Process Metric (17) Paper P15 presents four metrics describing the strength of verification applied to the security requirements and mechanisms of the system being measure: Coverage (M92), Depth (M101), Rigor (M234), and Independence of Verification (M137). Coverage is a categorical measure of the extent to which all security functionalities are examined during verification. Depth is a categorical measure of the portion of security functionalities that are examined during verification. Rigor is a categorical measure of the maturity of the security functionality verification process. Independence of Verification is a categorical measure of the extent to which security functionalities are evaluated by persons other than the implementers of the functionality.

Paper P52 includes a checklist of qualitative measures of process strength and risk, including CMMI Level (M66 ,P52), Development Risk (M112, P52), Environment Risk (M117, P52), and Project Management Risk (M202, P52).

Paper P61 describes Oracle’s philosophy and implementation of software development security metrics. Process-specific metrics they mention include ‘Number of security bulletins issued per year’ (M252, P61), ‘Volume of email correspondence with vulnerability handling team’ (M292, P61), ‘Use of (automated) tools’ (M282, P61) by the development team.

3.3.1.4 Incident Metrics (36)

We identified two subcategories of Project Management Metrics: Vulnerability Metrics, and CVSS Metrics. We now discuss each of these subcategories.

Vulnerability Metric (20) Twenty papers defined or used a count of vulnerabilities, ‘Vulnerability Count’ (M293, P14, P18, P19, P25, P36, P44, P45, P46, P53, P54, P56, P58, P60, P61, P62, P64, P67, P68, P69, P71). We found multiple theoretical and practical definitions of Vulnerability Count. In terms of theory, authors provided five definitions of ‘vulnerability’ in the seven papers that defined the term:

- an instance of a fault in the specification, development, or configuration of software such that its execution can violate an implicit or explicit security policy (P18, P60, P62, P71, citing Krsul [40].)
- ‘a defect, which enables an attacker to bypass security measures’ (P36, citing Pfleeger [74])
- ‘a weakness in the security system that might be exploited to cause loss or harm’ (P36, citing Schultz [85])
- ‘Vulnerability is a software defect that can be exploited to cause a security breach.’ (P69)
- an ‘unknown system characteristic’ (P63)

In terms of practical definitions of vulnerability, we identified five schemes used to count vulnerabilities:

- CVE counts from public vulnerability databases (NVD, OSVDB, Bugtraq) (P14, P36, P53, P60, P64, P67, P68, P69, P71)
- CVE counts from public vendor vulnerability databases (MFSA, RHSR) (P18, P25, P46, P60, P68)
- CVE counts from privately-curated vulnerability databases based on public sources (P46)
- Vulnerability counts from development team vulnerability databases (Cisco) (P44, P62)
- Vulnerability counts from analysis tools (P39, P58)

Paper P7 presents three date attributes for vulnerabilities, Vulnerability Discovery Date (M296, P7), the date on which the vulnerability is discovered, Vulnerability Patch Date (M301,

P7), the date on which the solution to the vulnerability is shipped, and Vulnerability Disclosure Date (M63, P7), the date on which the vulnerability is publicly announced.

While not made explicit, the status of a vulnerability as patched or un-patched can be inferred from the presence or absence of a Vulnerability Patch Date. Further the status of the vulnerability as being open or resolved can be similarly inferred.

Vulnerability Density (M294, P28, P36, P58) is a count of vulnerabilities normalized to the number of (thousands of) SLOC in the vulnerable code unit.

‘Structural Severity’ (M263, P64) is a qualitative, categorical metric indicating the vulnerability’s proximity to the attack surface.

‘Window of exposure’ (M307, P66) is the length of interval of time taken by the security operations team to patch a vulnerability once it has been disclosed.

‘Vulnerability-Contributing-Commit’ (M304, P71) indicates the presence of a commit in the version control repository that contributed to the introduction of a post-release vulnerability.

‘Vulnerability Discovery Rate’ (M297, P69) is a count of vulnerabilities per time unit (usually month).

‘Vulnerability Free Days’ (VFD) (M298, P14) is ‘the percent of days in which the vendors queue of reported vulnerabilities for the product is empty.’

CVSS Metrics (16) Eight papers used CVSS [55] metrics (P8, P13, P21, P41, P42, P53, P64, P70). CVSS metrics describe characteristics and impact of vulnerabilities, and so can be seen as a more detailed notion of vulnerability than a simple count. The CVSS metrics are categorized as Base, Temporal, and Environmental, representing characteristics of the vulnerability that are constant, that change over time, and that are associated with the environment in which the vulnerability was found. Details of the metrics can be found in Mell [55], and online ⁴.

3.3.1.5 Resource Metrics (34)

We identified metrics as part of the Resource Metrics category when the metric is primarily a measure of some cost or effort incurred in the course of security attacks or defense. We identified three subcategories of Resource Metrics: Cost Metrics, People Metrics, and Time Metrics. We now discuss each of these subcategories.

Cost Metrics (9) *We collected metrics related to financial concerns as well as attacker and defender effort, cost, and reward in the ‘Cost Metric’ category.*

We group measures of the value of an attack to an attacker and the cost of an attack to the defender (“One man’s loss is another man’s gain”) as ‘Attack Value’ (M24), for example

⁴<https://www.first.org/cvss/specification-document>

‘Damage Potential’ (P34), ‘Mean Failure Cost’ (P24), and ‘Annual Loss Expectancy’ (P43).

We group ratios of Attack Value to Attack Effort as ‘Attack Risk’, for example ‘Attackability’ (M21, P22), and ‘Security Risk Index’ (M21, P29).

We group measures of the non-monetary costs of an attack to the attacker as ‘Attack Effort’ (M16, P22), for example the number of steps required to complete an attack (M216, P26), or the ease with which a vulnerability is exploited (M16, P29, M488, P64) Where ‘Attack Effort’ is measured in monetary terms (e.g. M14, P26, M292, P34), we refer to it as ‘Attack Cost’.

Attack Cost Two papers define Attack Cost (P26, P37). Authors defined Attack Cost in the following ways:

‘a combination of everything the attacker must spend in order to successfully execute an attack (money, resources, time, effort, etc.)’ (P26)

‘Reflect the cost of T(Technique) in Attack Path. Let Cost(x) be the attack cost produced during the software Attack Path x. Attack Cost is quantifiable, and its unit is exp.’ (P37)

By analogy with attacks, we defined ‘Defense Effort’, ‘Defense Cost’, ‘Defense Value’, and ‘Defense Reward’ (ratio of value to cost/effort). Two metrics focused on defense, ‘Minimal cost for reduction of attacks’ (M212, P26), and ‘Return on Penetration Testing’ (M337, P43).

People Metrics (23) We identified five distinct classes of people metrics: Developer, Attacker, User, Project Manager, and Organization. ‘Developer Count’ (M107, P25, P53, P60) is a count of the number of developers who made changes to the software being measured. Authors defined Developer Count in the following ways:

NumDevs - ‘Files were changed by many developers’ (P25)

Number of Engineers - ‘absolute number of unique engineers who have touched a binary and are still employed by the company’ (P53)

NumDevs - ‘The number of distinct developers who changed the file’ (P60)

P53 defines ‘Number of Ex-Engineers’ as the ‘total number of unique engineers who have touched a binary and have left the company as of the release date of the software system.’ Meneely (P71) defines variants of Developer count, including ‘Number of distinct authors besides the commit author whose lines were affected by a given commit’, and New Effective Author, indicating the first time a developer makes a commit.

Paper P25 contains a set of developer activity metrics (M185-M194) that describe developer relationships through graphs. ‘Developer Risk’ (M111, P52) measures concern over developer

ability for a project. Similarly, ‘Software Project Management Experience’ (M251, P52) and ‘User Risk’ (M288, P52) measure expected strength and weakness of managers and users. ‘Social Engineering Resistance’ (M250, P29) measures how well users are able to resist attempts at social engineering. ‘Information Leakage’ (M139, P20) measures the degree of information transfer between developers on a software project.

Attackers are measured in terms of their capability (M28, P29), level of motivation (M29, P29, M30, P51), skill level (M31, P29, M31, P50), and speed (M32, P50). Two papers define attacker (threat agent) skill. Authors define Attacker Skill in the following ways:

A qualitative assessment of attacker skill (Low, Medium, High) (P77)

The expected value of a random variable in an item-response theory model (P92)

Organizations are measured in terms of the percentage of contribution to a piece of software’s development (M192, P53) and the percentage of the organization that has received security training (M191, P61).

Time Metric (2) Oracle Corporation measures ‘Time to close bug/vulnerability’ (M266, P61), a form of ‘Mean time to repair’. ‘Interval between violation and report’ (M151, P33) is the length of the interval between the time the vulnerability is discovered and the time the vulnerability is reported to the development team. ‘Attack Execution Time’ (M17, P50) is the length of time required to execute the attack being measured.

3.3.1.6 Defensibility Metrics (19)

We identified metrics as part of the Defensibility Metrics category when the metric is primarily a measure of the ease or difficulty with which a system is attacked. We identified two subcategories of Defensibility Metrics: Attack Surface Metrics, and Attackability Metrics. We now discuss each of these subcategories.

Attack Surface Metrics (9) *We collected measurements of attack surfaces in the Attack Surface Metrics category.* An ‘attack surface’ represents the paths in and out of a system (e.g. input and output parameters or fields), the data that travels those paths, and the code that protects the paths and the data ⁵.

Five of the selected papers used attack surface metrics (P11, P26, P34, P35, P64).

- A systems attack surface is the set of ways in which an adversary can enter the system and potentially cause damage. (P11)

⁵https://www.owasp.org/index.php?title=Attack_Surface_Analysis_Cheat_Sheet&oldid=156006

- A systems attack surface is the subset of its resources that an attacker can use to attack the system. (P11, P26, P34, P35, P64)

‘Sensitive Sink’ (M248, P45) indicates that execution of the code labeled as a sensitive sink may lead to harmful operations, making it a form of exit point.

‘Attack Graph Probability’ (M18, P10) is defined as ‘the intrinsic likelihood of an exploit e being executed, given that all the conditions required for executing e in the given attack graph are already satisfied.’.

Paper P67 defines ‘Exploitability Risk’ (M125), a qualitative measure of whether a vulnerability is accessible from a system’s entry points.

Paper P11 developed a formalism for measuring attack surfaces and an ‘Attack Surface Metric’ (M23, P11) representing the attack surface size adjusted for the risk of sharing the data resources exposed through the attack surface. The authors of P11 conducted a validation study of the metric in Paper P34.

Attackability (9) ‘Attackability’ (M27, P3) is the probability that an entity will be successfully attacked. ‘Attack Count’ (M15, P26) defines how many attacks on a system exist. The idea behind this metric is that the greater number of attacks available for a system, the less secure the system is. ‘Attack Prone’ (M26, P47) is the likelihood that a component will be attacked (measured in Paper P47 by the presence of previous vulnerabilities). Paper P7 defines a set of metrics describing the probability that a system is secure, has been compromised, and has, or has not, been repaired.

‘Vulnerability Index’ (VI) has three distinct definitions. In Papers P16 and P49, VI (M299) is the probability of a components vulnerability being exposed in a single execution multiplied by number of expected executions. In Paper P56, VI is a qualitative (categorical) assessment of a state of a system (be it a router, a server or a client), which can be normal, uncertain and vulnerable. In Paper P49, VI indicates ‘the percentage of vulnerability occurrences that have been removed from the corresponding artifact.’

3.3.1.7 Aggregate Metrics (18)

We identified metrics as part of the Aggregate category when the metric is a combination of other metrics that aggregates across metrics and dimensions to present an overview of some aspect of the measured entity’s security.

In the course of categorization, we identified a set of aggregates, combinations of other metrics designed to summarize the overall security state of the entity being measured. One of

the simpler aggregates, ‘CVSS Score’ (M102, P41, P42, P54) is defined as a ‘weighted average of CVSS impact (Confidentiality/Integrity/Access Impact) and exploitability (Access Vector, Access Complexity, Authentication) metrics.’. CVSS Score represents the severity of a single vulnerability. ‘Security Resource Indicator’ (M246, P28) combines numeric measures of the presence of four indicators: a security reporting email address, a security vulnerabilities list, a secure coding standard, and security configuration documentation. ‘Security factor requirements’ (M240, P2) combines metrics for Confidentiality, Privacy, Data Integrity, Authentication Non-repudiation, Auditing, and Intrusion detection to represent the overall security strength of a website. In contrast, ‘Expected Vulnerability’ (M124, P16) measures the overall security weakness of a system by aggregating measures of ‘Vulnerability Index’ (M299, P16) across the components of the vulnerable system. ‘Security Metric’ (M429a, P54) applies the same strategy as ‘Expected Vulnerability’, but aggregates CVSS Scores normalized to the number of vulnerability occurrences for the system being measured. Paper P1 represents the most substantial use of aggregates in the papers identified in our search. Beginning with the definition of classified attributes, the authors develop a system of metrics for the classes and methods that access the classified attributes, and then define a set of metrics representing various security properties in terms of the lower level definitions. For example, ‘Readability of Classified Attributes’ (RCA) (M219, P1) is composed of three lower level measures of attribute access via the system’s classes and instances. RCA is, together with ‘Readability of Classified Methods’ (RCM) (M220, P1), a component of ‘Fail-Safe Defaults’ (PFSD) (M127, P1). PFSD is defined as a security principle metric, representing the degree to which the system’s classified attributes are not accessible by default. A ‘Total Security Index’ (M275, P1) is computed using PFSD and six other aggregate security principle metrics representing the overall security of the software’s object-oriented design.

3.3.1.8 Security Requirement Metrics (16)

Paper P19 presents a series of metrics for measuring each stage of the software development lifecycle, including counts of security requirements, missing security requirements, and the ratio of security requirements to total requirements. Paper P57 presents metrics for security requirements, security use cases, and security misuse cases, together with a checklist for measuring system security requirements. Paper P52 includes an overall measure of Requirements Risk (M230, P52).

Authors defined Security Requirement Count in the following ways:

Total number of security requirements - number of security requirements identified during the

analysis phase of the application (M245, P19)

Ratio of security requirements - ratio of security requirements to the total number of requirements (M211, P19)

Number of omitted security requirements - number of requirements that should have been considered when building the application (M244, P19)

Ratio of the number of omitted security requirements - ratio of the number of security requirements not considered during analysis to the total number of security requirements identified during the analysis phase (M245, P19)

Number of excluded security requirements that ensure input/output handling (M180, P57) - defined by counting ‘No’ answers for a set of questions, e.g. ‘Is a specific encoding scheme defined for all inputs?’, ‘Are all the validations performed on the client and server side?’

Number of excluded security requirements that ensure session handling - defined by counting ‘No’ answers for a set of questions, e.g. ‘Is session identifier created on server side?’, ‘Is session identifier killed after a period of time without any actions?’

Alert Count Two papers, by the same authors, define static analysis tool Alert Count. Authors defined Alert Count in the following ways:

Automated static analysis (ASA) warnings (both security and non-security warnings) from two unnamed tools (P44)

ASA count of audited and un-audited null pointer, memory leak, buffer overrun alerts, and all of the reported by the FlexeLint tool.

In addition to raw Alert Counts, the metric was often normalized to SLOC, yielding Alert Density.

Security Test Metric Two papers defined and/or used ‘Security Test Case Count’ (P19, P57) counts of security test cases. Authors defined of Security Test Cases in the following ways:

‘represent security threats that the attacker might interact with to breach security and cause harm to the system’ (P57)

‘designed to detect security issues’ (P19)

Given the basic definitions, the authors define ratios of the security test case counts, relating them to, for example, total test case counts and failed/successful test cases.

Table 3.5 Metric Ids by Phase and Subject

Subject	Requirements	Design	Implementation	Testing	Operations	All
System	4	3	9	5	178	4
Source Code		5	98		10	
Component		3	65		6	8
Software version	3		12		17	
Design		9				
Requirements	8					
Repository			8			
Organization			1		5	2
Misuse case	8					
Commit			7			
Project						6
Security mechanism					4	
Service			1		2	
Model						2
User Account					1	

3.3.2 RQ1.2: What are the people, artifacts, and process elements being measured by software lifecycle security metrics?

As described above in Section 3.3.1.5, we identified five distinct classes of people metrics: Developer, Attacker, User, Project Manager, and Organization. We identified 15 distinct subjects, sorted by metric count: System (147 metrics), Source Code (77), Component (71), Software version (29), Design (8), Misuse case (8), Requirements (8), Organization (7), Commit (7), Project (6), Security mechanism (4), Service (3), Model (2), and User Account (1). We identified five phases, sorted by metric count: Operations (151), Implementation (145), Requirements (22), Design (18), and Testing (5), with 22 metrics deemed applicable to all phases. We present the metrics organized by the Phase in which they are measured, and the Subject which they measure in Table 3.5. Measures of running systems (System+ Software Version) (151) comprise the largest collection of metrics, followed by measures of Source Code. Measures of the software lifecycle and its non-source code artifacts are relatively few.

Table 3.6 Studies by Evaluation Technique

Reports from production Industry Case Study	P61 P11, P14, P16, P17, P25, P28, P32, P34, P35, P36, P39, P44, P45, P46, P47, P48, P49, P53, P58, P62, P67, P68, P69, P71
Academic Case Study	P1, P2, P3, P8, P15, P18, P22, P31, P42, P52, P54, P55, P60, P63, P64, P65, P67
Theoretical	P10, P12, P13, P19, P20, P26, P27, P30, P37, P40, P41, P50, P59, P6, P66, P7
Opinion Not Described	P24, P29, P43, P57, P9 P13, P23, P33, P38, P4, P5, P51, P56, P6, P70

3.3.3 RQ1.3: What validation approaches are used for software lifecycle security metrics?

We have classified each metric in terms of six validation approaches. Table 3.6 lists each validation approach and the papers applying that validation approach. A paper may appear more than once, as some of the papers apply multiple validation approaches. Broken down by Evaluation technique, the counts were: Industry User Study (116, 35%), Academic User Study (102, 31%), Opinion (35, 10%), Theoretical (64, 19%), Reports from production (6, 2%), Not Described (79, 24%).

3.3.4 RQ1.4 During what phases of the software development lifecycle are software lifecycle security metrics measured?

We have classified each metric in terms of five development phases, and report on the phases measured by the metrics in the selected papers. Table 3.7 lists each development phase and the

Table 3.7 Studies by Lifecycle Phase

Requirements	P19, P26, P27, P41, P57, P64
Design	P17, P18, P19, P30, P31, P65
Implementation	P1, P5, P9, P10, P12, P13, P18, P19, P22, P25, P28, P36, P44, P46, P47, P53, P55, P58, P60, P61, P62, P63, P64, P65, P67, P68, P69, P70, P71
Testing	P19, P43, P48
Operations	P2, P3, P4, P6, P7, P8, P11, P13, P14, P15, P16, P17, P18, P19, P22, P24, P25, P26, P27, P28, P29, P32, P33, P34, P35, P36, P37, P38, P39, P41, P42, P43, P44, P45, P49, P50, P51, P54, P56, P58, P59, P61, P62, P64, P65, P66, P67, P68, P69, P70
All	P20, P23, P40, P52

papers associated with that development phase. A paper may appear more than once, as some of the papers measure multiple development phases.

The most common lifecycle phase studied is Operations (50 studies), followed by Implementation (29 studies). The selected papers showed less emphasis on the Requirements (6 studies), Design (6 studies), and Testing (3 studies) phases of development. We identified four studies that defined metrics suitable for use across the development phases. For example 'Information Leakage' from P20 measures unwanted transfer of information between developers and project personnel during the development process.

Table 3.8 Lifecycle Phase by Metric Category

	Reqts	Design	Impl	Testing	Operations	All	Total
Aggregate	1		13		3	2	19
Defensibility		7		14	1	22	44
Implementation		8	131		3	1	143
Project Management		7	15		14	6	42
Resource			23	2	31	6	62
Security Properties		3	2		75	10	90
Security Requirements	16			2			18
Severity	6		10	1	83	2	102
Total	23	25	194	19	210	49	

3.4 Discussion

We reported on the results associated with each research question in Section 3.3. In this section, we make observations on trends in the results, based on our analysis of the selected papers and identified metrics.

3.4.1 Themes in Security Metrics

In this section, we present three themes we identified in the collected set of metrics.

3.4.1.1 Vulnerability Definitions and Counts Vary

The term vulnerability was defined in five different ways in the referenced papers (Section 3.3.1.4):

- an instance of a fault in the specification, development, or configuration of software such that its execution can violate an implicit or explicit security policy (P18, P60, P62, P71, citing Krsul [40].)
- ‘a defect, which enables an attacker to bypass security measures’ (P36, citing Pfleeger [74])
- ‘a weakness in the security system that might be exploited to cause loss or harm’ (P36, citing Schultz [85])
- ‘Vulnerability is a software defect that can be exploited to cause a security breach.’ (P69)
- an ‘unknown system characteristic’ (P63)

Vulnerability Count was the most-referenced metric; however, vulnerabilities were counted in five different ways, none of which depended on the definitions given for vulnerability. We repeat our findings on vulnerability counts from the results section here, sorted by paper count:

- CVE counts from public vulnerability databases (NVD, OSVDB, Bugtraq) (P14, P36, P53, P60, P64, P67, P68, P69, P71)
- CVE counts from public software vendor vulnerability databases (MFSA, RHSR) (P18, P25, P46, P60, P68)
- Vulnerability counts from development team vulnerability databases (Cisco) (P44, P62)
- Vulnerability counts from analysis tools (P39, P58)
- CVE counts from privately-curated vulnerability databases based on public sources (P46)

Use of the CVE records in the NVD database was the most common vulnerability counting technique. Massacci and Nguyen (P46) observe that the NVD is a partial record of vulnerability data, and that curation of all available sources of data, for example other vulnerability databases, reports from software vendors, and bug tracking data internal to the development team is necessary to provide accurate vulnerability counts.

The NVD CVE-based vulnerability counts are a record of public observations of security failures in software. Software development teams also track vulnerabilities internally and privately. Some projects (e.g. Google Chrome) handle vulnerabilities as a special case of defects, recorded and managed through the project’s bug tracker. Some projects provide an email address for sending vulnerability reports, but do not publish their tracking mechanism or vulnerability data. Distinguishing whether a project maintains a list of vulnerabilities may come down to inquiries to the project team.

Because the definition of vulnerability varies, the implementation of Vulnerability Count varies, and the availability of both varies, we recommend examining a given paper’s definitions of vulnerability and vulnerability count, particularly when comparing between papers.

We observe that while vulnerability count is the most-referenced metric, there is no standard definition or data collection procedure. We advise caution in comparing results across papers.

3.4.1.2 Many Metrics, Few Reused

Most (85%) security metrics have been proposed and evaluated solely by their authors. Less than 15% of metrics focus on the early stages of development, and less than 2% focus on testing. In our data, 40% of the metrics are not empirically evaluated. There are artifacts and processes that remain unmeasured. A potential research direction for security metrics would be for researchers to follow through on the evaluation and use of proposed metrics. **As seen by the broad range of metrics and the narrowness of those reported in more than one paper/by one set of authors, the field of software development security metrics has not settled on a baseline for measurement.**

3.4.1.3 Most Security Metrics are Subjective

The primary data sources for the metrics we surveyed are the subjective judgments of users, developers, managers, reviewers, and security researchers. In the metrics we surveyed, the two fundamental measurement operations are deciding on the assets to be protected (see Section 3.4), and deciding whether some behavior or outcome of the system is a violation of a security property, for example a vulnerability (see Section 3.4.1.1). Counting vulnerabilities depends on the counter's notion of some security property being violated. Even automated vulnerability counts (e.g. Paper P28) depend on the judgment of the implementers of the analysis tools used to raise warnings. Paper P1 developed a package of object-oriented security metrics founded on qualitative judgments about whether class and instance attributes are classified and/or critical. Paper P33 relies on qualitative assessments of security requirement and security mechanism criticality and strength. Attempts, like those of paper P22, to define metrics measuring how well code or components supports a security principle, e.g. 'Least Privilege' are a valuable contribution because they define how they ground the measured security principle and enumerate the set of subjective decisions required to measure the principle.

We see a need for developing and evaluating standards for assessing security needs at every point in the development lifecycle, analogous to Secure Coding Standards, e.g. Secure Requirements Standards, Secure Design Standards, and Secure Data Classification Standards. Where such standards exist, for example the Payment Card Industry Data Security Standards ⁶, we see opportunities for evaluation of the standard's effectiveness, development of test suites and fuzzers for thorough validation of software's compliance with the standards, and development of analogous standards for other types of data, e.g. social network graphs and password manager databases.

⁶https://www.pcisecuritystandards.org/pci_security/

Judgments made by users, developers, managers, reviewers, and security researchers about what constitutes an asset, and what constitutes a vulnerability are currently the primary source of security-related measurements in software development.

3.4.2 What To Measure?

Given the quantity and variety of metrics, one might ask where to begin when measuring security in software development. Lennon [45] suggests that security controls be implemented according to stakeholder priorities, with no more than 10 to 20 metrics in place to support manageability of the measurement program. We cannot speak directly to stakeholder priorities and goals, as they will vary, but we can illustrate an application of our collected metrics and their attributes. To illustrate the use of the data, and to set forth a baseline set of metrics, we select an example set of metrics drawn from our collected metrics, where the priorities are based on two criteria; the metric’s presence in two or more of our selected papers, and whether the metric’s evaluation was based on industry data. We include metrics in the example set if they are present in two or more papers, and they have been evaluated in an industrial context. We exclude Source Code Metrics, per our discussion in Section 3.3.1.2, with the exception of SLOC (M295), which is required to calculate Vulnerability Density (M227).

Table 3.9 presents the metrics referenced in two or more papers and evaluated in an industrial context from our results, organized by software development phase.

Our example set can be used as a baseline for discussion and adaptation to a given project, but the metrics are insufficient for measuring across the software development life cycle. Notably, the metrics in Table 3.9 apply only to the Implementation and Operations phases, as these phases have received the most attention by researchers in the literature we examined. To measure security concerns during the Requirements phase, we recommend considering the metrics presented in papers P41 and P64. To measure security concerns during the Design phase, we recommend considering the metrics presented in paper P31. To measure security concerns during the Testing phase, we recommend considering the metrics presented in papers P19, P43, and P48. **Selection of metrics for a given project is currently a matter of stakeholder preference.**

3.4.3 Fundamental Security Questions, revisited

We now return to the example ‘fundamental’ questions from the introduction (Section 1), and discuss the application of the metrics we have collected to answering the questions.

Table 3.9 Metrics referenced in two or more papers and evaluated in an industrial context

Name	MId	Paper Id	Category	Phase
Alert Count	M342	P44, P47	Implementation Metrics	Implementation
Commit Count	M522	P60, P68	Implementation Metrics	Implementation
CVSS Metrics	M518	P64, P67, P70, P13	Defensibility Metrics	Operations
Defect Count	M340	P19, P44	Project Management Metric	Implementation
Developer Count	M414	P25, P53, P60, P71	Resource Metrics	Implementation
Source Lines of Code (SLOC)	M295	P18, P28, P36, P44, P46, P47, P53, P58, P69	Implementation Metrics	Implementation
Vulnerability Density	M227	P28, P36, P58	Implementation Metrics	Implementation
Attack Effort	M292	P22, P29, P34, P37, P50	Resource Metrics	Operations
Attack Risk	M291	P22, P29, P34, P56	Resource Metrics	Operations
Attack Surface Metric	M289	P26, P34, P35	Defensibility Metrics	Operations
Attack Value	M290	P22, P34, P48, P43, P24	Resource Metrics	Operations
Security Resource Indicator	M231	P28, P58	Aggregate Metric	Operations
Vulnerability Count	M100	P14, P18, P25, P36, P44, P45, P46, P53, P58, P60, P61, P62, P68, P69, P71	Severity Metrics	Operations

3.4.3.1 Have we considered security in each stage of the software life cycle?

Metrics are available for each development phase we identified (Section 3.3.4, Table 3.7). However, the Requirements and Design phases have relatively few metrics.

3.4.3.2 What are the security risks to which our software is exposed?

Resource Metrics (Section 3.3.1.5) can be used by the development team and other stakeholders to assign economic values to the risks involved in the use of the software. Defensibility Metrics (Section 3.3.1.6) can be used to assess the level of risk inherent to the design and implementation of the software. Incident Metrics (Section 3.3.1.4) can be used as a direct measure of the risks that have occurred during the use of the software.

3.4.3.3 What are the security assurances we can give to users of our software?

The Security Properties Metrics (Section 3.3.1.1) can be used to characterize how security properties are implemented in software. The Project Management Metrics (Section 3.3.1.3) can be used by the development team and other stakeholders to assess the development process for attention to quality.

3.5 Limitations

If we have seeded our Quasi-Gold-Standard (QGS) with the wrong papers, we may have excluded relevant papers from our results. We drew our results from three search engines, ACM, IEEE, and Elsevier, limiting our selection of papers to what is available in their indexes. Our QGS scores were low for Elsevier, suggesting that we may have missed relevant papers.

While we attempted to be comprehensive in our search strings and result parsing, our approach may have missed papers. Limiting our search to the scholarly literature excluded existing standards as well as industry experience reports disseminated by other means.

Software development organizations may choose not to report whether they are using metrics, limiting our observations to discussion of the scholarly literature.

Our metric classification scheme reflects our own biases in the data elements and values selected for each metric. We mitigate this bias by drawing on previous work where applicable (e.g. Savola [82], the IEEE glossary [31], and Fenton [20]). Given the scheme, the choices made by individual raters are also subjective. We attempted to reduce bias by applying our two rater scheme.

Drawing inferences from the fields we classified depends on how accurately our choices match objective reality. Data elements we synthesized (Category, Measured Subject) are especially subject to this limitation, though we had two extractors check each metric-category assignment. We did not attempt a second approach, or a second set of extractors, to compare results, so our measures of validity need confirmation through replication.

3.6 Contributions of the Systematic Mapping Study

Our systematic mapping study identified 324 unique metrics for measurement of security-related elements of and concerns in the software development life cycle. We developed a set of categories and subcategories for the metrics that may offer guidance for metric selection and a basis for metrics discussions. We have observed that many metrics are focused on the characteristics of vulnerabilities in running software, and that relatively fewer metrics are available for measuring vulnerability prevention during the early phases of development.

At present, security is measured primarily in its absence, as signified by the proportion of metrics dedicated to describing and counting vulnerabilities in software. As a consequence of the focus on vulnerabilities, most metrics are measured during the Implementation and Operations phases, once the faults causing the vulnerabilities are embedded in software. We see a need for further research into how to measure the presence of security, and how to identify fault introduction opportunities in the early development phases, e.g. Requirements and Design.

Even the most-used, most cited security metric, Vulnerability Count, must be carefully examined for the means used to measure it, in terms of what constitutes a vulnerability, and what the inclusion and exclusion rules are for including a vulnerability in the count. The CVSS standard is a step in the direction of standardization, but leaves a great deal of room for variation because rater judgment drives scoring. If vulnerability counts are shared between stakeholders, all participating stakeholders should be familiar with the definitions used.

Our database of collected software development security metrics can be used as a guide to research needs in the field of software development security metrics.

3.7 RQ1.5: What software development security practices are used by software development teams?

In this section, we present the methodology and results for our review of software development security practices.

3.7.1 Methodology

We develop our list of software development security practices to investigate through content analysis [84], which is a grounded theory data collection technique [23]. In brief, content analysis calls for establishing a research question, selecting material to analyze, designing a “coding frame” with which to analyze the material to answer the research question, piloting and refining the coding frame, analyzing the material, and reporting on the results.

To capture security practice use in a consistent way at the level of actions needed within roles on a development team, we developed a template to record each security-related action described in our four sources, translating the practices identified into a consistent format. The security practice template is as follows: *< Role >< Verb >< Artifact(s)Affected > guidedby :< Artifact(s)Referenced >, during < Phase >.*

- The Role element indicates the role played by the person applying the practice (“who”).
- Verb names the action taken by its user (“what”).
- Artifact is used as a generic term for the objects and information sources used or created during software development.
 - Artifact(s) Affected is a list of artifacts that are created, or changed by the practice (“where”).
 - Artifact(s) Referenced is a list of artifacts that are used to inform decisions by the user as they apply the practice (“why”).
- Phase indicates when in the development process the practice is applied (“when”).

To illustrate how examples from our sources align with the template, we draw on filled-in template examples from three of our sources:

- Source: SDL “NULL out freed memory pointers in code” is templated as Role: Developer, Verb: Apply, Artifact Affected: Source Code, Artifact Referenced: Secure Coding Standard, Phase: Development
- Source: SAFECode “Avoid String Concatenation for Dynamic SQL Statements” is templated as, Role: Developer, Verb: Apply, Artifact Affected: Source Code, Artifact Referenced: Secure Coding Standard, Phase: Development
- Source: BSIMM “Use Coding Standards” is templated as Role: Developer, Verb: Apply, Artifact Affected: Source Code, Artifact Referenced: Secure Coding Standard, Phase: Development

We classified each of these examples as an instance of the practice “Apply Secure Coding Standard”.

We use the template as our guide in classifying the practices identified in our sources. We then use the set of records generated by classifying the practices for identifying the main relationships between practices, artifacts, and roles.

We are focused on the activities of software developers and testers, and so we include activities where developers and testers perform the activities, or use the results of practices performed by others. For example, BSIMM’s “Develop an operations inventory of applications” is important, but software development teams are not necessarily tasked with operational tasks, and the practice is also necessary for organizations that do not develop their own software. We set the following inclusion criteria for selecting practices from our sources:

- Practices employed by software developers and software testers
- Practices that produce inputs for practices used by software developers and software testers

We set the following exclusion criteria for excluding practices from our sources:

- Examples illustrating practice use.
- Language, environment, and network specifics

We applied these criteria by classifying each practice in terms of the role of the person to apply the practice, and in terms of the artifact to which the practice is applied. Where a practice did not apply to a software development role or artifact, we marked it as excluded.

The classification scheme described above was applied to each of four security practices sources, BSIMM, SDL, OWASP, and SSAP.

3.7.2 Practice classification results

We identified 462 practices and practice statements in our sources. The complete set of source statements and their classifications is available online ⁷. We excluded 85 practices as being out of scope for our focus on software development, leaving 377 practices to classify. Through our classification, we identified 16 core practices that, in combination with roles, verbs, phases, and artifacts, were sufficient to describe the 377 source practices we identified. Table 3.10 lists the practices, counts for appearances in each source, and a total count, sorted by frequency of appearance across all sources

⁷ anonymized

Table 3.10 Security Practices By Source

SPEFPractice	Source				Total
	BSIMM	CLASP	MS SDL	SAFECODE	
Apply Secure Coding Standards	10	2	68	9	89
Perform Security Review	23	0	21	0	44
Perform Security Testing	10	3	20	4	37
Document Technical Stack	14	6	4	7	31
Apply Security Tooling	11	1	12	2	26
Apply Security Requirements	7	11	7	0	25
Track Vulnerabilities	16	0	8	0	24
Apply Threat Modeling	9	4	5	1	19
Provide Security Training	13	2	3	1	19
Improve Development Process	14	0	0	0	14
Perform Penetration Testing	9	0	2	1	12
Apply Data Classification Scheme	11	1	0	0	12
Publish Operations Guide	4	4	2	0	10
Apply Security Principles	0	1	4	3	8
Monitor Security Metrics	1	4	0	0	5
Publish Disclosure Policy	0	2	0	0	2
Excluded	68	0	14	3	85

To aid concision and simplify data collection for SP-EF, we excluded practices mentioned less than 10 times in our sources, namely ‘Apply Security Principles’, ‘Monitor Security Metrics’, and ‘Publish Disclosure Policy.’ Table 3.11 lists the SP-EF practices in software development phase order with their associated roles, verbs, and artifacts.

By aligning the actions the development team takes with the effects of those actions on project deliverables, we can measure how security practice use affect development outcomes.

3.7.3 Discussion

Our list of software development security practices has face validity based on the development procedure we followed. In the coming chapters, we turn to assessing content and construct validity through use of the security practices list in case studies and surveys of software development projects.

3.7.4 Limitations

We built our list of software development security practices based on four published sources. Our sources may bias the practices toward those used by large organizations building desktop and web applications. Applying the survey and model in small organization contexts, and in embedded systems contexts may reveal different or additional practices. The projects we surveyed span a range of sizes and domains, suggesting that the practices we identified are used across a spectrum of projects.

Table 3.11 SP-EF Security Practice Descriptions

Name	Description
Perform Security Training	Ensure project staff are trained in security concepts, and in role-specific security techniques.
Apply Data Classification Scheme	Maintain and apply a Data Classification Scheme. Identify and document security-sensitive data, personal information, financial information, system credentials.,
Apply Security Requirements	Consider and document security concerns prior to implementation of software features.
Apply Threat Modeling	Anticipate, analyze, and document how and why attackers may attempt to misuse the software.
Document Technical Stack	Document the components used to build, test, deploy, and operate the software. Keep components up to date on security patches.
Apply Secure Coding Standards	Apply (and define, if necessary) security-focused coding standards for each language and component used in building the software.
Apply Security Tooling	Use security-focused verification tool support (e.g. static analysis, dynamic analysis, coverage analysis) during development and testing.
Perform Security Testing	Consider security requirements, threat models, and all other available security-related information and tooling when designing and executing the software's test plan.
Perform Penetration Testing	Arrange for security-focused stress testing of the project's software in its production environment. Engage testers from outside the software's project team.
Perform Security Review	Perform security-focused review of all deliverables, including, for example, design, source code, software release, and documentation. Include reviewers who did not produce the deliverable being reviewed.
Publish Operations Guide	Document security concerns applicable to administrators and users, supporting how they configure and operate the software.
Track Vulnerabilities	Track software vulnerabilities detected in the software, and prioritize their resolution.
Improve Development Process	Incorporate "lessons learned" from security vulnerabilities and their resolutions into the project's software development process.

Chapter 4

A Security Practices Evaluation Framework

4.1 Introduction

We propose the Security Practices Evaluation Framework (SP-EF), a measurement framework to enable empirical data collection for security practice use and outcomes in software development.

While researchers have empirically evaluated software development practices for their security benefits, for example, in requirements engineering [79], design patterns [91], threat modeling [88], static and dynamic analysis [86], code review [90], and testing [4], these studies have approached each practice in isolation, leaving open the question of how a combination of practices applied to a project affects its security outcomes. By including measurements for a representative set of software development security practices, we construct SP-EF to enable measurement of combinations of practices as used on software development projects.

Establishing cause and effect is a difficult matter. For example, prominent statisticians including R.A. Fisher criticized the initial studies correlating smoking with cancer [77]. They argued that multiple causes and multiple confounds might offer alternative explanations for the correlation identified between smoking and cancer [48]. Wasserman [96] observes that the eventual scientific consensus was formed after numerous observational studies adjusting for many confounding variables, laboratory experiments showing smoking to damage lung cells, and causal links found in randomized animal studies. Wasserman [96] summarized the state of practice for scientifically credible observational studies in terms of three criteria:

- a plausible scientific explanation for the existence of a causal relationship,
- studies controlling for plausible confounding variables, and
- results replicated across many studies,

We anticipate that assessing the effect of software development security practice adherence on security outcomes in industrial contexts will require addressing each of these criteria: plausible causal relationships, confounding variables, and replication.

To support detailed analysis for how security practices affect security outcomes, we represent each element of the framework in a model of the software development lifecycle, enabling study of the relative importance of each security practice in software development, and measurement of the validity and reliability of the framework.

To address plausible causal relationships between security practice adherence and security outcomes, we define an analysis model, Security Outcomes Theoretical Model (SOTM), in Section 4.4 for how adherence to a set of security practices affects security outcomes for a software development project. We define a common adherence measurement scheme for the 13 security practices included in SP-EF, presented in Table 3.11. We represent each element of the SP-EF framework in SOTM, enabling study of the relative importance of each security practice in software development, and measurement of the validity and reliability of the framework.

To address confounding variables, we measure and include in our analysis model in Section 4.4 the metrics we identified in our literature review as having been evaluated more than once in industrial contexts (Section 3.4.2), for example the software’s size and purpose. These metrics primarily describe the software project’s context, and are included in SP-EF’s Context Factors.

To support replication, we draw from the example of XP-EF [98] measurement framework, a set of repeatable measures for assessing the degree to which a team holds to XP practices, enabling replication and meta-analysis of studies. We emulate the organization and intent of XP-EF in the construction of SP-EF, while adapting the content to focus on security practice adherence rather than Extreme Programming. SP-EF provides guidelines for data collection of context factors, practice adherence metrics, and outcome measures. Practice adherence metrics are a means of characterizing the degree to which a practice is used on a project. SP-EF includes objective and subjective metrics for measuring practice adherence. Outcome measures characterize the software’s security performance.

In this chapter we present the SP-EF data elements with rationales for why the data element was chosen, and the SOTM used to assess the impact of each data element when SP-EF is applied to empirical data.

4.2 Context Factors

We reviewed the the XP-EF list of context factors with one of the XP-EF authors, and kept context factors that were mutually agreed as being relevant to security. We then augmented the context factors list with the metrics we identified in our literature review as having been evaluated more than once in industrial contexts (Section 3.4.2). The complete list of metrics is available in the guidebook [63].

4.3 Practice Adherence

Software development teams may adhere to security practices in varying degrees. We would like to measure the degree to which a team adheres to each security practice. Researchers in technology adoption have suggested that “perceived ease of use” (“Ease of use”) and “perceived usefulness” (“Effectiveness”) are central to technology adoption by individuals [16, 92].

Venkatesh [92] identified two further factors as central to technology adoption by individuals within organizations: facilitating conditions and social influence. Venkatesh defines “facilitating conditions” as “[t]he degree to which an individual believes that an organizational and technical infrastructure exists to support use of the system” and social influence as “[t]he degree to which an individual perceives that important others believe he or she should use the new system.”. The complete Venkatesh model, the Universal Theory of Acceptance and Use of Technology (UTAUT), applies these factors, and moderating factors, such as age and gender, to predict user intentions of technology use and actual user technology use. Multiple applications of UTAUT across domains, e.g. those reviewed by Williams [99] have confirmed that positive changes in the UTAUT model factors are predictive of user intentions and actual use.

We adapt the UTAUT model to our purpose of measuring security practice adherence. In the present study, we measure individual perceptions by presenting Likert scales with the following values: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree. For our Ease metric, we ask “Rate your agreement with ‘This Practice is easy to use’”. For our Effectiveness metric, we ask “Rate your agreement with ‘This practice assists in preventing and/or removing security vulnerabilities on our project’”. We investigate “facilitating conditions” through asking a question about training in each practice: “Rate your agreement with ‘I have been trained in the use of this practice’”. To estimate social influence, the degree to which an individual perceives important others believe that the practice should be used [92], we compute our “Influence” metric as the ratio of survey participants who report Usage of a practice to the total number of participants. Given that we have access to only the survey participants

for each team rather than complete team information, we do not compute “Influence” in the present study.

UTAUT addresses questions of user perception of the effectiveness of technology adoption. In SP-EF, we also wish to support managerial goals for the efficiency of technology adoption. We introduce a fifth practice adherence metric, “Effort”, to support investigations of practice cost and developer productivity for security practice use. For each practice, we ask “How much time, on average, does it take to apply this practice each time you apply it?”, with the following scale values: 15 minutes or less, 15-30 minutes, 30 minutes-1 hour, 1-4 hours, 4-8 hours, 1-2 days, 3-5 days, More than 5 days, Not Applicable.

4.3.1 Outcome Measures

The goal of security effort in software development is to produce software that will maintain the security properties desired by users and owners, making the ideal outcome a measure of the absence of security breaches, exploits, and flaws. Conceptually (see Section 4.4.1.4), security outcomes may be positive (no exploited vulnerabilities), or negative (vulnerabilities). Parallel to Dijkstra’s [buxton1969software] observation that testing can be used to show the presence of bugs but never to show their absence, the absence of discovered vulnerabilities may be reflective of low discovery effort rather than indicative of low actual vulnerabilities present in software. Negative measurements of security outcomes must be considered in the light of context factors such as time the software has been in use, and adherence factors such as the long-term use of security testing. At present, SP-EF contains outcome measures for negative security outcomes. The SP-EF outcome measures depend on records of defects and vulnerabilities. At a minimum, counts of defects and vulnerabilities should be taken. Where possible, detailed records including the full set of defect and vulnerability attributes should be maintained. The complete list of metrics is available in the guidebook [63].

4.4 The Security Outcomes Theoretical Model (SOTM)

To evaluate how security practices affect security outcomes, we need to account for other influences on security outcomes. We propose the SOTM model of the factors influencing security outcomes in software development to enable assessment of how varying security practice use affects those outcomes, while accounting for software usage’s effect.

The Common Criteria (CC) introduction [56] lists a set of **security concepts** (concepts bolded) and relationships, summarized as follows:

- Owners value **Assets** and impose **Countermeasures** to reduce **Risk**.
- Threat Agents give rise to **Threats** that affect **Assets** and increase **Risk**.

Considering the CC concepts in the context of software development and use, we propose a model to capture the distinction between the risk to Assets caused by Threats (Usage Risk) and the risk caused by the software that manages the Assets (Development Risk) when measuring practice Adherence’s effect on security Outcomes.

We have theorized relationships between measurement variables and each construct in our model. For example, we theorize that code size and code churn metrics influence Development Risk. We present our list of measurements and the construct relationships in the ‘Construct’ column for each metric in Table 4.1.

4.4.1 Structural Model Constructs

In this section, we define the Development Risk, Usage Risk, Adherence, and Outcomes constructs, and the relationships we expect between each construct.

4.4.1.1 Development Risk

Development Risk (CC **Risk** potential) represents the characteristics of the software produced by the development team that are associated with defects and vulnerabilities. In the case of software vulnerabilities, for example, high code churn [86] and defect-prone languages [78] are two examples of software characteristics that have been correlated with vulnerabilities. See Catal and Diri [13] and Morrison et al. [65] for two examples of surveys of vulnerability-related metrics.

4.4.1.2 Usage Risk

Usage Risk (CC **Assets**) represents the characteristics of the software’s purpose and usage context that are associated with attacker interest. One component of attacker interest is the value of the assets managed by the software. For example, we conjecture that software tracking valuable or sensitive data, such as Personally Identifiable Information (PII) or credit card data is more likely to be attacked than software tracking baseball scores. As another example, attacker control over a machine enables the machine’s participation in a botnet, making the number of machines on which a piece of software runs a consideration in evaluating the software’s Usage Risk.

4.4.1.3 Adherence

Adherence (CC **Countermeasures**) represents the efforts the team takes to prevent and discover vulnerabilities. When effort is reported for security practice evaluations in the software engineering literature, it is typically reported in terms of time spent. For example, Austin et al. [3] measured the time required to apply penetration testing tools, static analysis tools, and testing to locate vulnerabilities, and Dowd et al. [17] reports security code review rates of 100-1000 SLOC per hour.

In the information technology literature, researchers have developed ‘technology adoption’ models for how technology users adhere to new technologies. Davis’ [15] Technology Acceptance Model (TAM) posits perceived usefulness and perceived ease of use as constructs central to a user’s intention to use a given technology, and TAM has been applied and reviewed extensively, e.g. Venkatesh [92]. Venkatesh [92] examined TAM and seven related models and extensions to TAM, and proposed the Unified Theory of Acceptance and Use of Technology (UTAUT) model. UTAUT retains the TAM constructs, while adding Social Influence and Facilitating Conditions as primary constructs driving user intention to use a given technology.

4.4.1.4 Outcomes

The Outcomes (CC **Risk** realized) construct represents indications of the failure or achievement of security associated with a piece of software over the course of the software’s life cycle.

At present, counting vulnerabilities is the most common means of measuring security in software [65]. However, as pointed out by Meneely [60], vulnerabilities measure breaches in security rather than the presence of security. Where possible, analogues to the reliability engineering measurement of mean time between failure should be measured for the assessment software security.

4.4.2 Structural Model Relationships

We hypothesize that the four constructs are related as follows:

- **H1** Usage Risk is associated with negative Security Outcomes
- **H2** Development Risk is associated with negative Security Outcomes
- **H3** Development Risk is inversely associated with Practice Adherence

For example, a carefully designed, written, and tested piece of widely-used software that manages financial data (high Usage Risk, low Development Risk, e.g. Bitcoin with 22 CVEs ¹) might have poorer Outcomes than a less well written baseball scores program used by a few hundred users (low Usage Risk, high Development Risk, no CVEs reported) because attackers expend more effort on the software managing financial data. We would expect Adherence to be correlated with Usage Risk, as teams adopted security practices in proportion to the security needs of their software, its usage context, and their users. In practice, for example in the case of the CloudBleed example from Section 1, users (especially attackers) sometimes surprise development teams in the uses of their software, unexpectedly increasing the software’s Usage Risk out of proportion to the team’s Adherence.

Figure 4.1 depicts the constructs and their relationships. Each circle in the figure represents a construct, modeled as a ‘latent variable’. We model the constructs using latent variables to indicate that our measures for each construct are aggregates of the measurement (observed) variables [37, 8].

Directed edges from circles to other circles, for example the arrow from Usage Risk to Outcomes in Figure 4.1, represent a source latent variable’s effect on the target latent variable. SEM estimation generates parameter estimates for each edge. The sign and magnitude of the parameter estimate on the edge between variables represent how changes in a source variable covary (or correlate, for standardized estimates) with changes in a target variable in the direction of the parameter estimate’s sign and of a magnitude comparable to other parameter estimates

Dual arrows between circles/constructs, for example between Adherence and UsageRisk in Figure 4.1, represent covariances between the constructs, implying that a relationship exists, but that the direction of influence is not specified. Dual arrows starting and ending at the same variable (not shown in Figure 4.1) indicate the variable’s observed variance.

Each square in Figure 4.2 represents a measurement variable associated with each construct. Directed edges (single arrows) from circles to squares, for example from DevelopmentRisk to SLOC as shown in Figure 4.2, represent that a construct ‘is measured by’ a measurement variable relationship. That is to say that the effect of the construct can be measured by the measurement variable. The parameter estimate on these relationships represents the sign and magnitude of the relative contribution of the measurement variable to the construct value. These parameters and their magnitudes are sensitive to the combination of measurement variables included by researchers in their measurement of a latent variable. We present the list of measurements, and instructions for collecting them, for each construct in the SP-EF measure-

¹https://www.cvedetails.com/vulnerability-list/vendor_id-12094/Bitcoin.html

Figure 4.1 Structural Model Overview

Figure 4.2 Graphical display of lavaan syntax for $\text{DevelopmentRisk} = \sim$ Theorized measurement variables from Table 4.1

ment guidebook [61].

4.4.3 Measurement Model: Metrics, Variables, and Relationships

Through literature review [65] and analysis [67, 66], we have developed a set of measurements that we expect to capture security-related constructs for software development. In Table 4.1, we name each data element, give our hypothesis about its relationship to the structural model construct, and cite a rationale for the data element’s presence. The metrics associated with Development Risk and Usage Risk are termed Context Factors. The metrics associated with Adherence and Outcomes are termed Measures.

Table 4.1 Model Context Factors, Measures, and Hypotheses

Metric		Effect	Construct	Rationale
Language		influences	Development Risk	Ray et al. [78] and Walden et al. [94] found small but significant effects of programming language on software quality. Zhang [103] identifies language as a key context factor.
Operating System		influences	Development Risk	
Domain		influences	Development Risk	Different risks are associated with different software domains [97, 33]
Product Age		increases	Development Risk	Kaminsky et al. [34] and Morrison et al. [64] have found evidence of code age effects on the presence of vulnerabilities.
Source Lines of Code (SLOC)		influences	Development Risk	Source code size is correlated with vulnerabilities [86], [1], [11], [14].
Churn		increases	Development Risk	Zhang [103] identifies SLOC as a key context factor. Code churn is correlated with vulnerabilities [86].
Team Size		influences	Development Risk	Shin et al. [86] and Zimmermann et al. [105] found correlations between team size and vulnerabilities.
Number of Machines		increases	Usage Risk	(Proposed) The market for machine time on botnets suggests that the number of machines a piece of software runs on increases the software's desirability to attackers.
Number of Identities		increases	Usage Risk	(Proposed) The market for personal identities and credit card information suggests that the number of identities a piece of software manages increases the software's desirability to attackers.
Number of Dollars		increases	Usage Risk	(Proposed) The amount of financial resources a piece of software manages increases the software's desirability to attackers
Source Code Availability		influences	Usage Risk	While Anderson [2] argues that attack and defense are helped equally by the open vs. closed source decision, we collect this data to enable further analysis.
CIA Requirements		increases	Usage Risk	Explicit confidentiality, integrity, and availability requirements for a piece of software imply a higher level of Usage Risk for the software [55].
Team Location		influences	Adherence	(Proposed) Kocaguneli [38] reports on the debate over the effect of team location on software quality, collecting data on team location supports study of its effect.
Methodology		influences	Adherence	Different risks are associated with different software methodologies [97, 33]
Apply Data Classification Scheme		increases	Adherence	(Proposed) Identifying data in need of protection supports reducing Development Risk [67].
Apply Security Requirements		increases	Adherence	(Proposed) supports reducing Development Risk[ref Riaz, etc] [67].
Apply Threat Modeling		increases	Adherence	(Proposed) Identification and analysis of threats supports reducing Development Risk [67].
Document Technical Stack		increases	Adherence	(Proposed) Understanding and controlling platform and dependency characteristics supports reducing Development Risk [67].
Apply Secure Coding Standards		increases	Adherence	(Proposed) Avoiding known implementation errors supports reducing Development Risk [67].
Apply Security Tooling		increases	Adherence	(Proposed) Automated static and dynamic security analysis supports reducing Development Risk [67].
Perform Security Testing		increases	Adherence	(Proposed) Explicit validation of security requirement fulfillment supports reducing Development Risk [67].
Perform Penetration Testing		increases	Adherence	(Proposed) Exploratory testing of security properties supports reducing Development Risk [67].
Perform Security Review		increases	Adherence	McIntosh et al. [53] observed lower defects for highly reviewed components. Meneely et al. [59] observed lower vulnerabilities for components with experienced reviewers.
Publish Operations Guide		increases	Adherence	(Proposed) Documenting software security characteristics and configuration requirements supports reducing Development Risk [67].
Track Vulnerabilities		increases	Adherence	(Proposed) Incident recognition and response supports reducing Development Risk [67].
Improve Development Process		increases	Adherence	(Proposed) Adoption and adaptation of security tools and techniques based on experience supports reducing Development Risk [67].
Perform Security Training		increases	Adherence	(Proposed) Development team knowledge of security risks and mitigations supports reducing Development Risk [67].
Vulnerabilities		represent	Outcomes	Vulnerabilities are, by definition, a negative security outcome, e.g. [1].
Defects		represent	Outcomes	Zhang [103] identifies defect tracking as a key context factor.

4.5 Contribution

In SP-EF, we have produced a set of measurements of software development projects that have a theoretical basis for being predictive of software security attributes. In SOTM, we have built an assessment framework for SP-EF, allowing estimation of how much impact each measurement has on software security outcomes, and allowing estimation of the importance of each security practice to a software's security outcomes.

Novel aspects of SP-EF include:

- A representative set of 13 software development security practices.
- A scheme for measuring practice adherence in a way that allows practices to be compared with each other.
- An explicit model, SOTM for how security outcomes are affected by security practice adherence and confounding variables. SOTM makes explicit what is being measured, and what is not being measured, allowing researchers to assess the quality of a replication, and to develop numeric estimates for the effects of each modeled variable and relationship.
- The notion of Usage Risk, distinct from Software Risk, as a group of context factors relevant to measuring security outcomes in software development.

We have defined SP-EF, a set of context factors, practice adherence measures, and security outcome measures. However we have not verified that we can collect all of the required data for a single software project. In the next chapter, we will apply the framework to an industrial software project as a means for evaluating both the framework and the project.

Chapter 5

Measuring Security Practice Use in Software Development

In this chapter ¹, we present a case study of a small IBM software development project as test of whether the SP-EF measurements can be gathered in the context of a software development project, as well as a study of how security practices are applied during software development in a large organization.

One component of addressing *RQ2* ‘Can the measurements affecting software development security outcomes be measured on software development projects?’ is to verify that we can collect the necessary measurements for a single project. Reflecting this need, our research question for the chapter is **RQ2.1**: Can the complete set of SP-EF measurements be collected for a software development project?

We collected empirical data from three perspectives: qualitative observations, a survey of the team members, and text mining of the team’s development history. We observed the team directly and report our qualitative observations of the team’s security practice use, guided by the SP-EF subjective practice adherence measures. We surveyed the team, based on the SP-EF subjective practice adherence measures, for the team’s perspective on their security practice use (see Appendix C for the questionnaire). We mined the team’s issue tracking history for security practice use. By comparing the observations, we build up a picture of the team’s security practice use and outcomes. We present a set of lessons learned.

The rest of this chapter is organized as follows: . Section 5.1 gives an overview of our case study methodology and data collection procedures. Section 5.2 presents the industrial project

¹Portions of this chapter appear in Morrison et al. [66]

study results. Section 5.3 discusses the case study findings. Section 5.4 presents a list of lessons learned. Section 5.5 presents our study limitations. Finally, Section 5.6 presents our conclusion.

5.1 Methodology

To assess our SP-EF data collection methods and to increase confidence in our findings, we triangulate, collecting data in three ways; through qualitative observation, survey, and text mining of the team’s issue tracker. We worked with the project staff and read through the project documentation for qualitative evidence of the security practices, based on the SP-EF subjective practice adherence measures. We conducted a survey of the team, using the SP-EF practice adherence survey(Appendix C). The survey contains demographic questions (e.g. role, time on project), four questions aligned with the SP-EF subjective measures for each SP-EF security practice, and open-ended questions allowing survey participants to express their views.

We obtained objective SP-EF practice adherence measures for the team by applying a basic text mining technique, keyword counting, to the project’s issue tracking records, as of month 31 of the project. The text mining classification procedure is available as an R package, linked from the SP-EF website [63]. To develop an oracle for assessing the performance of the text mining, we read and classified the set of issue tracking records described above according to the guidelines for identifying the presence of SP-EF practices. We compute recall and precision for the mined objective measures, compared to the manual oracle.

5.2 Industrial Case Study Results

In this section, we present the results of the case study. Project observation and data collection was conducted during Summer 2014 and 2015, through embedding the researcher as a developer on the project team. We take the case to be a typical example of security practice use at IBM; the software is not security-focused, and the target environment is typical for the cloud environment. We studied 31 months of project history, from project inception through one year after public release, representative of the bulk of the development effort, and indicative of the transition from development to maintenance. We had access to the team’s issue tracker, wiki, source code repositories, as well as discussions with the team, on which we base our reported results.

5.2.1 RQ2.1: Can the complete set of SP-EF measurements be collected for a software development project?

We now present the results of SP-EF data collection for the project's context factors, practice adherence and outcome measures.

5.2.1.1 Context Factors

We present the SP-EF context factors to describe the project in which the security practices are applied:

- Confidentiality Requirement: Low
- Integrity Requirement: High
- Availability Requirement: High
- Dependencies: WebSphere Application Server, OpenSSL
- Domain: Web Application Utility
- Number of Identities: 0
- Language: Ruby, Java, Javascript
- Number of Machines: 10000
- Methodology: Agile
- Operating System: Unix
- Product Age: 31 months
- Source Code Availability: Closed Source
- Team Location: Distributed
- Team Size: Two managers, fourteen developers, two testers, one technical writer.

Figure 5.1 presents an overview of how four context factors changed over the course of the project: code churn, source code repository commits, number of committing developers, and total source lines of code (SLOC). We expect that project context factors affect security practice choice and usage, but testing the nature and strength of those relationships will require comparison with data collected from other projects.

Figure 5.1 Context Factors

5.2.1.2 Practice Adherence

In this section, we present our findings on the use of each security practice from three data collection perspectives; qualitative observation, survey, and text mining, summarized in Table 5.1. Based on internal project membership lists and management approval, we sent links to the survey to the 18 active team members. We left the survey open for three weeks (during month 28 of the project), providing a reminder email three days before we closed the survey. Eight team members responded (44% response rate); five developers, two requirements engineers, and one “Other”. The respondents averaged three years of experience. The team uses RTC work items to record planned tasks as well as reported defects. We collected work items for the most recent 19 months of the project. Figure 5.2 presents the team’s responses to the four survey questions for each practice, where the practices are sorted from greatest agreement (or greatest frequency) to least. The numbers following the practice names are the number of respondents who answered for the practice and the number of total respondents, e.g. all respondents (8/8) used “Track Vulnerabilities”. The “Mining Counts” section of Table 5.1 presents the number of occurrences of references to each practice in the work items, as counted manually (“Oracle”) and by the keyword counting script (“Mined”). The “Performance” section of Table 5.1 presents recall, precision and the F1 scores for keyword counting compared against the manually-generated oracle. We now present our findings on the use of each of the 13 security practices.

Table 5.1 Practice Adherence Comparison Table

Practice SP-EF Security Practice	Researcher Observation			Mode of Survey Response				Mining Counts		Performance	
	Pres	Freq	Prev	Freq	Ef	Ea	Tr	Oracl	Mined	Precision	Recal F1
Apply Data Classification Scheme	Yes	Annual	Low	Annual	A	SA	A	5	6	0.00	0.00 N/A
Apply Security Requirements	Yes	Weekly	High	Weekly	A	SA	N	21	204	0.03	0.33 0.06
Apply Threat Modeling	Yes	LT Annual	Low	Weekly	A	SA	D	7	21	0.00	0.00 N/A
Document Technical Stack	Yes	Monthly	High	Quarterly	SA	SA	A	65	744	0.05	0.60 0.09
Apply Secure Coding Standards	Yes	Daily	High	Daily	A	SA	N	9	554	0.01	0.44 0.02
Apply Security Tooling	Yes	Weekly	Medium	Daily	SA	A	D	9	184	0.03	0.67 0.06
Perform Security Testing	Yes	Weekly	High	Weekly	A	SA	N	348	658	0.50	0.94 0.65
Perform Penetration Testing	Yes	Annual	Low	Monthly	N	SA	N	2	5	0.40	1.00 0.57
Perform Security Review	Yes	Monthly	High	LT An- nually	A	SD	N	31	47	0.21	0.32 0.25
Publish Operations Guide	Yes	Monthly	Low	Quarterly	A	SA	N	42	359	0.04	0.31 0.07
Track Vulnerabilities	Yes	Weekly	High	Daily	SA	SA	N	36	192	0.11	0.58 0.18
Improve Development Process	Yes	Monthly	Medium	Monthly	SA	A	N	102	8	0.00	0.00 N/A

Pres = Presence - Practice is used.; Freq = Frequency - How often is practice used?;
Prev = Prevalance - Percentage of team using the practice.
Ef = Effectiveness; Ea = Ease; Tr = Training
SD = Strongly Disagree; D = Disagree; N = Neutral, A = Agree; SA = Strongly Agree

Apply Data Classification Scheme (ADCS) IBM employees receive annual computer-based training courses in security principles, including material on how to identify and classify data appropriately. The product does not directly manage user or company data, although applications using the product may do so. According to the survey, ADCS is the least frequently used practice, used no more than quarterly by 5 of 6 respondents. However, ADCS is used daily by one of the six respondents. ADCS also ranked low for ease of use, utility, and training. The text mining oracle indicates ADCS is the second-least-used practice after PPT, and keyword-counting had poor performance locating ADCS. Combining perspectives, we assess that ADCS is used very little by the team.

Apply Security Requirements (ASR) The team uses Rational Team Concert ² (RTC) to manage its work, including “work items” for each feature, defect, and vulnerability. Team discussions during iteration planning and daily scrum meetings, including security-related requirements, are captured in the work items and their comments. The work items are used to drive development and testing efforts. The team logged 898 stories over the course of the project, with 81 of these mentioning security as a topic (9%). According to the survey, ASR is used weekly, is viewed positively by the team, although they are neutral on their training. The text mining oracle indicates ASR is the second-least-used practice after PPT, and keyword-counting had poor performance. Combining perspectives, we assess that ASR is used by the team, reflected more

²<http://www-03.ibm.com/software/products/en/rtc>

by the qualitative observations and survey than by text mining.

Apply Threat Modeling (ATM) The second author participated in several sessions with the team’s technical leadership to develop a threat model for the product. The threat model is expressed as a set of meeting notes identifying the components of the system and constraints to be maintained by management and engineering. According to the survey, ATM is used weekly and is viewed positively by the team, although they indicated negative views on training. The text mining oracle indicates ATM is rarely referenced in the work items (7 references), and keyword-counting had poor performance locating ATM. Combining perspectives, we assess that creating the threat model is a rare event, and that use of the threat model, while present, may not be recorded in the artifacts we studied.

Document Technical Stack (DTS) Developers on the team maintain a wiki for the project components and development environment. A team member is tasked with keeping the team current on patches required for components used in the product and development environment. According to the survey, DTS is used quarterly and is viewed positively by the team on all measures. The text mining oracle indicates DTS is referenced in the work items (65 references), and keyword-counting had 60% recall but low (5%) precision, overstating the presence of DTS by a factor of 10 (744 references). Combining perspectives, we assess that the team uses DTS, but our text mining technique overstates its presence.

Apply Secure Coding Standards (ASCS) IBM has internal coding standards for each of these languages, and code is reviewed by peers and technical leads. The build process includes automated standards checks for each language. According to the survey, ASCS is used daily and is viewed positively by the team, although they indicated neutral views on training. The text mining oracle indicates ASCS is rarely referenced in the work items (9 references), and keyword-counting had 44% recall and low (1%) precision, overstating the presence of ASCS by a factor of 10 (554 references). Combining perspectives, we assess that the team uses ASCS, but our text mining technique understates its presence.

Apply Security Tooling (AST) The project’s automated test suite includes security-focused verification tests. The team’s security person performs static and dynamic analysis on the product by running AppScan ³. According to the survey, AST is used daily and is viewed positively by the team, although they indicated negative views on training. The text mining oracle indicates AST is rarely referenced in the work items (9 references), and keyword-counting had 67% recall but low (3%) precision, overstating the presence of AST by a factor of 20 (184 references). Combining perspectives, we assess that the team uses AST frequently, but it is not

³<http://www-03.ibm.com/software/products/en/appscan>

mentioned in work items.

Perform Security Testing (PST) Each change to the product must be accompanied by a unit test. In addition, two quality assurance (QA) developers maintain a System Verification Test (SVT) automated test suite. QA uses SVT to exercise the product for its defined use cases, to apply additional validation and verification tests as they arise. SVT must pass before a new release of the product. According to the survey, PST is used weekly and is viewed positively by the team, although they indicated neutral views on training. The text mining oracle indicates PST is referenced more than any other practice in the work items (348 references), and keyword-counting had 94% recall and 50% precision, the best performance achieved in our text mining. Combining perspectives, we assess that the team uses AST frequently, with unanimous agreement by qualitative, survey, and text mining results.

Perform Penetration Testing (PPT) The product is evaluated annually by penetration testing teams external to the team but internal to IBM. PPT had Annual frequency in qualitative observation and a mode of Monthly for the survey response. PPT was the least common practice (2 references) according to both the text mining oracle and keyword-counting (5 references). According to the survey, PPT is used monthly and is viewed positively for ease of use, although the team indicated neutral views on utility and training. The text mining oracle indicates PPT is referenced less than any other practice in the work items (2 references), and keyword-counting had 100% recall and 40% precision, overstating the presence of PPT. Combining perspectives, we assess that PPT is applied to the project, but the team participates in PPT only indirectly.

Perform Security Review (PSR) Coding standards are automatically checked at each (weekly) build. Non-author team members review every source code change. The team conducts annual security assessments based on IBM secure development assessment questions. Code changes made for security reasons are infrequent. According to the survey, PSR is used less than annually, and is viewed positively for utility, although the team indicated negative views on ease of use, and neutral views on training. The text mining oracle indicates PSR is referenced infrequently (31 references), and keyword-counting had 21% recall and 32% precision, overstating the presence of PSR. Combining perspectives, we assess that the team has a disciplined code review process, and that security concerns are relatively infrequent during review.

Publish Operations Guide (POG) The team documents how to configure, administer, and use the product, and revises the documentation in conjunction with its releases of the product ⁴. According to the survey, POG is used quarterly, and is viewed positively by the team, although they indicated negative views on training. The text mining oracle indicates POG is referenced

⁴<https://github.com/cloudfoundry/ibm-websphere-liberty-buildpack/tree/master/docs>

infrequently (42 references), and keyword-counting had 31% recall and 4% precision, overstating the presence of POG. Combining perspectives, we assess that POG is used infrequently by the team.

Track Vulnerabilities (TV) IBM's Product Security Incident Response Team ⁵ (PSIRT) operates a company-wide clearinghouse for vulnerability information related to IBM products. The PSIRT team forwards vulnerability reports, and an assigned team member creates RTC "work items" to track team decisions about the PSIRT vulnerability through their resolution. According to the survey, TV is used daily, and is viewed very positively by the team, although they indicated neutral views on training. The text mining oracle indicates TV is referenced infrequently (36 references), and keyword-counting had 58% recall and 11% precision, overstating the presence of TV (192 references). Combining perspectives, we assess that TV is used frequently by the team, but text mining work items is a poor source for evidence of TV.

Improve Development Process (IDP) Team members discuss opportunities to improve the effectiveness and efficiency of the development process. The infrastructure - RTC, wiki, automated builds, automated test suites - embeds much of the development process knowledge in artifacts rather than in the memory of individual team members. The team uses the infrastructure to record and apply refinements to the development process. According to the survey, IDP is used monthly, and is viewed positively by the team, although they indicated neutral views on training. The text mining oracle indicates IDP is referenced about 11% of the time in the work items (102 references), but keyword-counting did not correctly classify an instance of IDP. Combining perspectives, we assess that IDP is used by the team.

Perform Security Training (PST) IBM employees receive annual computer-based training courses in security principles, including material on how to identify and classify data appropriately. According to the survey, the team were positive about their training in ADCS and DTS, negative about their training in ATM and AST, and neutral about their training in all other practices. We did not text mine for training. Combining perspectives, we assess that the organization applies PST, but the team seeks additional training.

5.2.1.3 Outcome Measures

The team logged 249 defect items over the course of the project, with 21 of these having potential vulnerability concern (8.4%). In terms of code changes made because of vulnerabilities, 1 critical patch to a vulnerability was applied to the software during the period measured, yielding a vulnerability density of approximately 1%, low compared to other projects measured using

⁵<http://www-03.ibm.com/security/secure-engineering/process.html>

Figure 5.2 Survey Results

SP-EF⁶.

5.3 Discussion

The team applies all 13 SP-EF practices, though it does not apply all practices equally. We did not find security practices not described in SP-EF.

In comparing our qualitative observations, survey responses, and text mining results, we found both close matches and wide differences. Matches included PST and ADCS. PST was strongly attested to by each type of measurement, although the counts of mentions in the work item records suggest that the practice is mentioned even more often than “Weekly” as reported by both observation and survey mode. Our conjecture is that practices may be mentioned more (or less) often in work items than they are used by individuals on the team. ADCS is lightly used according to all three measures. Differences included ATM and PSR. For ATM, both researcher observation and text mining showed low incidence, however survey responses indicated weekly effort. Further investigation is required to assess the difference.

In our survey results, we found variance in how often team members apply the practices. Five of the 13 practices had the same mode from survey responses as the qualitative observation. Everyone answered that they apply TV at least weekly, but PST was not used by all team members, and those who did varied in their frequency of use from daily to less than annually. We conjecture that project roles influence security practice use and frequency of use. Similarly, we found variance in team survey responses for their training. Everyone agreed or strongly agreed that they had been trained for TV, however no one rated PPT stronger than neutral. We conjecture that the type and recency of training, and the frequency with which a practice is applied influence the survey responses. ATM showed the largest difference between the survey mode (“Weekly”) and qualitative observation (“Less Than Annually”). We based our qualitative observation on recognition that the team had developed a threat model in a series of sessions early in the project. The higher frequency reported by team in the survey results is based on their applying, rather than creating, the threat model. The team rated 9 of the 13 practices as “Neutral” for training, suggesting that opportunities for increasing the frequency and depth of training in specific practices.

To rank the practices according to the combined survey responses (acknowledging the methodological problems with doing so), we compute a survey adherence metric as follows; assign the values -2, -1, 0, 1, and 2 to the Likert values Strongly Disagree... Strongly Agree,

⁶ For example, SP-EF VDensity measurements of phpMyAdmin ranged between 10% and 30%, and Firefox ranged between 0.47% and 2%.

multiply by the number of responses for each value, and sum these scores for the three Likert questions for each practice. The top three practices for this team were TV (tie), ASCS (tie), and DTS. When, additionally, accounting for frequency of use (assigning “number of times per year” for each of the frequency values, e.g. 12 for Monthly, 50 for Weekly, 200 for Daily), the top three practices were TV, ASCS, and AST. We do not find it surprising that TV ranked highest, given that a person is assigned to tracking, and that IBM provides organizational support for ensuring vulnerabilities are identified. What may be more surprising is that ASCS ranks next, perhaps reflecting that attention to quality during construction is valued by the team. We plan to conduct further case studies to measure the reliability of the survey responses and the factors that influence them.

The text mining measures had low precision, and varied from low to perfect recall. Performance tended to increase in proportion to the number of references to the keywords in the work items. Improvement in the text mining technique is required before text mining can be relied on for practice identification. Achieving improvement is of value in both having an objective means for observing practice use, and for handling projects with too much to manually classify. We will consider at least three improvements for future work: refining the SP-EF keywords, identifying project-specific keywords, and applying alternative text-mining methods.

5.4 Lessons Learned

Beyond the specific practice adherence data collected and reported, we synthesized several lessons learned from our observation of the team:

- *Security is not just a team effort, it is an organizational effort* The project team receives significant support from IBM as an organization, through the provision, monitoring, and enforcement of corporate documentation, process, and security standards, and through alerts to vulnerabilities in software that the project depends on.
- *Text mining is not a panacea* Our text mining technique is basic, and many more sophisticated techniques are available and should be explored, for example classifiers built using Naive Bayes, Support Vector Machines, or Random Forests. However, even the work items oracle sometimes overstates and sometimes understates practice use in comparison to the other measures of practice use. Variations in how developers record descriptions of practice use will affect the performance of any text mining technique. Identifying security practices through text mining may require practice-specific data and techniques to capture the variation in how security practices are described in project artifacts.

- *Watch your dependencies* The code the team develops is not the only code to be concerned with; changes in the project’s technical stack may require accommodation. For example, the project’s software stack includes OpenSSL, which required three patches during the period measured.
- *Investment in quality assurance is investment in security* The project has disciplined planning, coding, testing, and release procedures, full time testers, an automated test suite, and a team member assigned to monitoring of security issues. In our observation, one of the effects of the investment in process was to yield security-related information from formal channels that led to potential problems being discussed and resolved by the team through informal communications like voice, whiteboard, and instant messaging.

5.5 Limitations

The team in our study is a convenience sample based on the researcher’s location within IBM’s corporate hierarchy. The list of survey participants was rooted in team membership over time, but the team manager had the final say on who received the survey. These factors may bias the reported results. As a single project case study, we do not have a basis for establishing external validity. The single team within a single organization, and the small sample of participants, work against being able to generalize our results. However, the multiple perspectives of the studies’ data collection establishes a baseline researchers can use when assessing future teams. Our sources may bias the practices toward those used by large organizations building desktop and web applications. Applying the survey and model in small organization contexts, and in embedded systems contexts may reveal different or additional practices. Within the contexts of our sources, the face validity of the practices was confirmed by our empirical observations on the practices.

5.6 Contributions of the Measurement Case Study

We conducted a case study of security practice adherence on a small industrial software development team by applying the SP-EF security practices and adherence measures. We found agreement between the researcher and surveyed team views of security practice use on the project, and evaluated the effectiveness of automated means of assessing practice adherence. We identified use of all of the practices specified in SP-EF by one or more survey participants.

Studying security practice adherence in a single case study does not generalize to other

software development projects. In the next chapter, we survey a set of open source projects to further explore our measures of security practice adherence.

Chapter 6

Surveying Security Practice Use in Software Development

6.1 Introduction

In this chapter ¹, we report on a security practice adherence survey of open source development teams.

To further study RQ2 'Can the measurements affecting software development security outcomes be measured on software development projects?' in an open source context, we ask:

- **RQ2.2:** What software development security practices are used by software development teams?
- **RQ2.3:** Does security practice adherence, as measured by Ease of use, Effectiveness, and Training, correlate with software development security practice use?

To address the research questions, we surveyed 11 security-focused open source software development teams for their security practice adherence, based on the survey instrument presented in Chapter 5.

Our contributions include:

1. Empirical evaluation of security practice adherence in a set of open source software development projects.

¹Portions of this chapter appear in Morrison et al. [67]

The rest of this chapter is organized as follows: Section 6.2 presents our research questions and methodology. Section 6.3 presents the study results. Section 6.4 discusses implications of the results. Section 6.5 presents our study limitations. Finally, Section 6.6 presents our conclusion.

6.2 Methodology

In this section, we present our study methodology, subject selection, and data collection procedures.

6.2.1 Study Methodology

We conducted our study according to the following plan:

- To study projects that could be expected to apply security practices, we obtained developer email addresses from the published data of open source projects with the following characteristics:
 - project vulnerabilities are recorded in the National Vulnerability Database (NVD)²
 - version control system data is available
 - issue tracking system data is available
 - developer email archive data is available
- We sent the survey (Appendix C) to the developer email lists identified in the previous step. Each survey invitation included a brief explanation of our study and its goals, and a drawing for a \$25 Amazon gift card, to encourage participation.
- We collected the survey responses, limiting our final data set to those participants who had indicated consent to use their data, analyzed the responses according to the plan laid out below for each research question, and report the results here.

6.2.2 RQ2.2: What software development security practices are used by software development teams?

Metrics: Count of users for each practice. Frequency of usage for each practice, as measured by the scale of the survey responses. Qualitative practice responses reported by survey participants.

²<http://www.nvd.com>

In terms of validity, the SP-EF list of software development security practices has face validity based on the development procedure we followed (Section 3.7.1). However, even if our list is valid, we would expect that not all teams use all practices, and that the teams may use practices we have not identified. We defined a set of survey questions to measure variation in the use of practices within teams. To measure practice “Usage”, we include a survey question “How Often Do You Engage in the Following Activities?”, listing each of the 13 practices, with the following scale for responses: Daily, Weekly, Monthly, Quarterly, Annually, Less than Annually, Not Applicable. To assess whether our list of practices could be improved, we include a survey question inviting participants to suggest revisions or additions to the practices.

For each practice, we present user counts, a histogram of the Frequency scale and compute the mode (most commonly occurring Usage scale value). We summarize and report participant comments that suggest changes or additions to the practices.

6.2.3 RQ2.3: Does security practice adherence, as measured by Ease of use, Effectiveness, and Training, correlate with software development security practice use?

Metrics: Ease, Effectiveness, Training, Frequency, and Effort of each practice, as measured by the scales of the survey responses.

Following typical practice in evaluating UTAUT models. eg. Williams [99], we apply Structural Equation Modeling [37] (SEM) to our adapted model, using the data collected from the survey to test the following hypotheses:

- RQ2.3H1: Ease of use affects frequency of use of software development security practices. *Null: Ease of use is unrelated to frequency of use of software development security practices.*
- RQ2.3H2: Effectiveness affects frequency of use of software development security practices. *Null: Effectiveness is unrelated to frequency of use of software development security practices.*
- RQ2.3H3: Training affects frequency of use of software development security practices. *Null: Training is unrelated to frequency of use of software development security practices.*

We report the covariances and p-values for each hypothesized relationship in our practice adherence model.

Figure 6.1 Security Practice Adherence Measurement Model

6.3 Results

In this section, we present the findings of our investigations.

6.3.1 Practice classification results

By aligning the actions the development team takes with the effects of those actions on project deliverables, we can measure how security practice use affect development outcomes.

6.3.2 Survey Subjects and Responses

We selected Transport-Level Security (TLS) implementations: BouncyCastle, GnuTLS, mbedTLS, OpenSSH, and OpenSSL with the expectation that their teams would apply security practices in the course of their work. Our assumption is that TLS implementations exist to provide secure communications, so their teams would be sensitive to security issues in how the software is produced. We augmented the list with applications that were awarded Linux Foundation Common Infrastructure Initiative Best Practice badges ³, namely BadgeApp, Bitcoin, Node.js, and php-MyAdmin. Finally, we added the Firefox browser project to represent large-scale projects where

³<https://bestpractices.coreinfrastructure.org/>

Table 6.1 Survey Invitations and Responses, by Project

Project	Sent	Started	Completed
BadgeApp	5	1	1
Bitcoin	153	2	1
BouncyCastle	9	0	0
Firefox	1492	8	8
GNUTLS	30	0	0
mbedTLS	30	1	1
Node.js	78	1	1
OpenSSH	17	0	0
OpenSSL	155	0	0
OpenWISP	3	0	0
phpMyAdmin	24	1	1
Other	0	125	18
Total	1996	139	31

the development team pays attention to security. We present the surveyed projects, invitations and responses by project in Table 6.1.

We sent 1,996 surveys to developers on the listed projects, with 181 unreachable email addresses, 3 duplicate emails, 139 surveys started, and 31 surveys completed. The five questions about each of the 13 practices were required, but all demographic questions were optional. The 25 participants who indicated experience averaged 6.1 years of experience. The 26 participants who specified primary roles included Developers (17), Project Management (3), Security (3), Testing (1), a Build Administrator (1), and a Documentation/Technical Writer (1).

6.3.3 Research Question Results

In this section, we present the results for our research questions.

RQ2.2: What software development security practices are used by software development teams?

We present the frequency of use of each practice as reported by the survey participants in Figure 6.2. In our data, one or more participants reported daily use of each of the 13 security practices we presented. The two practices most often reported as daily practices were “Apply Secure Coding Standards” (14/31, 45% reporting daily use) and “Track Vulnerabilities” (13/31, 42% reporting daily use). At the other extreme, 42% (13/31) of respondents indicated that “Publish Operations Guide” was not applicable.

Figure 6.2 Security Practices Usage Frequency

In response to the open questions, participants made several important points on changes or additions for the survey practices.

- *“I would emphasize that many of these security practices are also general software development best practices.”*
- *“This might be subsumed by ‘Document Technical Stack’ and ‘Apply Threat Modeling’ but monitoring all of your open source dependencies and all of their dependencies for security advisories—not just patches—is important. ‘Apply Threat Modeling’ should be a daily task, reading information sources for early warning of new threats.”*

Additional practices suggested by participants include the following:

- *“Host source code publicly - Support cross-platform testing, code review by other developers with other standards ”*
- *“Reduce footprint - Focus the tools on doing one specific task well to avoid complexity introduced by multiple, conflicting requirements.”*
- *“Establish software requirements - Establish and document, library and system requirements to support the project.”*
- *“Support upgrade path - Provide safe and idempotent upgrade path for updates and releases.”*
- *“Mitigate potential security risks through design and implementation choices - Minimize security risks by choosing a language and frameworks that are less likely to introduce security issues, minimize the attack surface of your implementation by privilege separation, prepare appropriate mechanisms for timely delivery of security updates.”*
- *“Monitor ongoing trends in application security for new practices/techniques”.*
- *“Continuous Integration”*
- *“Fuzz Testing”*

One participant suggested removing “Perform Security Training”, explaining that *“Classroom knowledge delivered via lecture is useless at best. Experiential knowledge and mentorship through hands on experience is the only way to learn.”* One participant suggesting removing “Perform Security Review”, suggesting that testing is a more effective use of resources.

RQ2.3: Does security practice adherence, as measured by Ease of use, Effectiveness, and Training, correlate with software development security practice use?

We measure ease of use, effectiveness, and training using the questions, scales, and data from the survey. We measure practice use via the frequency and effort questions, scales, and data from the survey. Our structural model [37] of the constructs and their relationships to each other and to the measurement model for our study is presented graphically in Figure 6.1.

We collected a total of 31 sets of completed participant responses to the questions, where each participant's set of responses represents an observation. SEM calls for, as a minimum rule of thumb, 10 observations per parameter to be estimated. With a minimum of 69 parameters to be estimated (13 practices x 5 measures + 4 latent variables), we need no fewer than 690 observations. The quantity of data we collected is insufficient to conduct a SEM analysis.

In place of a full SEM analysis, we ran a linear regression on the modeled relationships, using the participant responses. To represent usage as our dependent variable, we converted frequency of use and effort from ordinal to ratio scales and multiplied frequency of use (instances of practice use per year) by effort (hours per instance of practice use) for each observation, yielding annual hours of practice use. We convert frequency to a ratio scale by treating each ordinal answer as 'number of times used per year', according to the following translation: Daily=260, Weekly=52, Monthly=12, Quarterly=4, Annually=1, Less than Annually=0.5, Not Applicable=0.

We convert effort to a ratio scale by averaging each ordinal answer range as 'number of hours per practice application' according to the following translation: 15 minutes or less=0.125, 15-30 minutes=.375, 30 minutes-1 hour=.75, 1-4 hours=2, 4-8 hours=6, 1-2 days=12, 3-5 days=32, More than 5 days=50, Not Applicable=0.

We regressed on usage (frequency*effort) as our dependent variable, and the three independent variables, ease, effectiveness, and training. Ease had a statistically significant relationship with usage for "Strongly Disagree" (545.83, p-value ≤ 0.01) and "Agree" (258.5, p-value 0.02). Effectiveness had a statistically significant relationship with usage for "Neither Agree nor Disagree" (292.25, p-value ≤ 0.01). Training had a statistically significant relationship with usage at an alpha of .10 at "Strongly Disagree" (-223.45, p-value 0.04), "Disagree" (-242.13, p-value 0.04) and "Agree" (-261.13, p-value 0.02). While not statistically significant, "Agree" (-133.95), and "Strongly Agree" (-49.26) suggest a trend where training increases the usage of practices (or decreases the disuse of practices).

6.4 Discussion

In this section, we present discussion of the results from the previous section. **RQ2.1: What software development security practices are used by software development teams?** As measured by survey participant responses, we found evidence for each of the 13 security practices we identified in our literature search. Usage varied widely between practices, and between project and participant use of each practice. In Table 6.2, we present the practices with their mean usage and the standard deviation of their usage, where usage is hours of use annually, as described in Section 6.3. “Apply Threat Modeling” and “Perform Penetration Testing” each have mean usage over 700 hours annually, while “Apply Data Classification Scheme”, “Perform Security Training”, and “Publish Operations Guide” each have less than 100 hours of mean annual usage.

Further investigation is required to interpret the meaning of usage. Low mean usage may indicate, for example, either that a practice is little-used, or that our practice names and descriptions may not be clear to survey participants. However, “Track Vulnerabilities” has the fourth-lowest mean usage, but seems likely to be both understood by participants and frequently used. We posit that low mean usage for frequently used practices may indicate that familiarity with the practice may make its application go more quickly than less familiar practices.

Participant comments suggest the possible addition of three practices. One participant suggested “Host source code publicly” to support cross-platform testing, and external review. We agree with the notion, but expect that such a practice will not be universal, as closed-source software remains economically viable. We speculate that escrow services, holding closed-source code for external inspection, will be required, in some situations, for closed-source programs that support critical infrastructure.

A second participant suggested “Reduce footprint - Focus the tools on doing one specific task well to avoid complexity introduced by multiple, conflicting requirements.”. We believe this suggestion generalizes to “Reduce Attack Surface”. In our current decomposition of practices, attack surface-related material is subsumed by the current set of practices (e.g. “Apply Security Metrics”, “Document Technical Stack”). However, every project has an attack surface, whether it is identified as such or not, and it may be worth identifying the attack surface management as an explicit practice.

A third participant suggested “Mitigate potential security risks through design and implementation choices”. We cover mitigation through implementation choices in “Document Technical Stack”, but distribute mitigation through design choices across “Perform Threat Modeling” and “Apply Security Requirements” because a separate design and associated documentation

is not universal among software development projects.

We observe that the participant suggestion to remove “Perform Security Training” came with recommendations for alternative forms of training, suggesting to us that the practice is of value but that varying forms of training should be considered in preparing team members and running projects. We weigh the participant suggestion to remove “Perform Code Review” in favor of further testing against the evidence of previous work showing that review is a more efficient use of resources than testing [50].

We take the 13 practices we identified to form a reasonable starting point for software development security practices.

RQ2.3: Does security practice adherence, as measured by Ease of use, Effectiveness, and Training, correlate with software development security practice use?

Our regression analysis of the practice usage data collected via the survey confirmed statistical significance for the theoretical relationships among the Ease of use, Effectiveness, Training, and Usage constructs predicted by UTAUT. Increases in training are reflected in increases in Usage. However, UTAUT also predicts positive relationships for Ease of use and for Effectiveness on Usage. While we should primarily conjecture that the negative relationships may be an artifact of the limited amount of data collected, and the significant variance in usage (e.g. 6.2), there may be a more complicated story to tell.

We re-ran the regression analysis broken down for each practice’s data, and report the slope (Positive or Negative, statistical significance indicated by “*”) and whether the relationship had statistical significance for the Ease-Usage and Effectiveness-Usage relationships in Table 6.2. Four practices have the theoretically-predicted relationships (e.g. “Document Technical Stack”), and five practices do not reach statistical significance for the relationships (e.g. “Improve Development Process”). As an example supporting theory, the highest use of “Perform Penetration Testing” was with an Ease of use of “Strongly Agree”.

We focus on the four practices that vary from theory and show statistical significance for both relationships: “Apply Security Tooling”, “Apply Secure Coding Standards”, “Track Vulnerabilities”, and “Perform Security Review”, and observe that while Effectiveness, as predicted by theory, varies positively with Usage for each practice, Ease of use varies negatively with Usage for each practice. We conjecture that the fact that highest Ease of use-Usage correlation for “Apply Security Tooling” with an Ease of use of “Strongly Disagree” may be a signal that successful implementation of security tooling is not easy. Alternatively, survey participants may find regular use of security tooling to be an impediment to their perceived progress. More generally, successful practice use may be marked by greater difficulty in practice application. Further data collection and analysis must be performed to draw conclusions about the relationships

Table 6.2 Practice Adherence Metrics Usage

Practice	N Users	Mean	SD	Ease-Usage	Effectiveness-Usage
Apply Threat Modeling	27	796.7	2367.7	Negative*	Positive
Perform Penetration Testing	28	741.6	2665.1	Positive*	Positive
Document Technical Stack	27	589.7	2148.3	Positive*	Positive*
Apply Security Requirements	28	558.1	2055.5	Negative	Positive
Improve Development Process	28	519.5	2121.5	Negative	Negative
Perform Security Testing	28	192.2	456.5	Negative	Positive*
Apply Security Tooling	29	184.6	429.4	Negative*	Positive*
Apply Secure Coding Standards	29	168.4	326.2	Negative*	Positive*
Track Vulnerabilities	29	152.7	204.8	Negative*	Positive*
Perform Security Review	30	122.3	167.2	Negative*	Positive*
Apply Data Classification Scheme	27	55.0	148.9	Positive*	Positive*
Perform Security Training	28	32.1	73.6	Positive	Negative
Publish Operations Guide	25	21.9	48.8	Positive	Positive*

between the constructs, and to establish the validity and reliability of the adherence metrics.

Toward a measure of practice adherence To fit the UTAUT model, we have treated Usage as a dependent variable in the present work’s analysis. We also view Usage as being of a kind with the independent variables in our analysis. All four measures, Usage, Ease of use, Effectiveness, and Training are correlated with a higher-level construct of Practice Adherence. For example, one participant observed that “Apply Threat Modeling should be a daily task, reading information sources for early warning of new threats.”. We conjecture that each practice has ideal values for each of the measures we have considered, moderated by the context of the project, and that comparing current values to empirically-validated “ideal” values may serve as actionable adherence metrics for software development security practices. For example, lower or higher than ideal usage can be addressed by discussing, or requiring, changing the frequency of use with the team. Lower than expected Ease of use can be addressed through, for example, examination and refactoring of work practices, and through training. Lower than expected Effectiveness can be addressed through examining practice use, and, possibly, discontinuing use of the practice. Low Training for a practice can be addressed through increasing the availability of Training. We plan further work to evaluate the Practice Adherence construct in terms of the dimensions of Usage, Ease of use, Effectiveness, and Training.

6.5 Limitations

We built our list of software development security practices based on four published sources (See Section 3.7.1). Our sources may bias the practices toward those used by large organizations building desktop and web applications. Applying the survey and model in small organization contexts, and in embedded systems contexts may reveal different or additional practices. The projects we surveyed span a range of sizes and domains, suggesting that the practices we identified are used across a spectrum of projects.

We have a single item to measure each of our practice adherence constructs (Ease of use, Effectiveness, Training). Typically, surveys use multiple items to establish a score for each construct. Extending the survey to ask a variety of questions about each practice adherence measure would strengthen measurement, however lengthening the survey will lengthen the completion time. The current survey required 10-15 minutes to complete by the participants who finished it, within the target set for the current study's design.

A survey of software development teams on their software development security practices puts participants in a position of having to answer questions about their performance at work on a sensitive topic. We would expect biases to influence participant answers in ways that we have not anticipated. Discovering, assessing for, and correcting for these biases will require further data collection and analysis both through the survey, and by other means, such as interviews and literature review.

Our empirical data is collected via a survey. The teams in our study are a convenience sample based on the research access to full project data.

As a second test of the list of practices and their adherence measures, we have limited basis for establishing external validity. The small sample of participants, and the small sample of teams within a work against being able to generalize our results. However, this study establishes a baseline researchers can use when assessing future results.

6.6 Contributions of the Survey Study

In this chapter, we found empirical evidence for the use of the SP-EF security practices through a survey of open source development teams. We conducted a survey of security practice adherence in open source development projects. We found empirical support for the use of the SP-EF security practices. We found that Training has a positive, statistically significant correlation with Usage, suggesting that investment in training supports practice usage. In principle, Training and the other adherence measures can be used by teams to guide practice selection and application.

For example, if a team identifies that most members indicate that they have not been trained in some practice used by the team, the team could invest in training in that practice. We have developed, and can make available, a security practice adherence survey instrument, and a research infrastructure for surveying software development team security practice use, collecting and analyzing data from the teams, and reporting on and interpreting the results of the analysis, enabling teams to monitor their software development security practice adherence. Apart from training, our survey data show that further data collection is necessary to confirm or reject the adherence measures.

In the previous two chapters, we have collected security practice adherence data through a case study and a survey. However, we do not have empirical evidence for whether the SP-EF framework and its underlying SOTM model meet the goals of observing and measuring security effort and security outcomes in software development. In the next chapter, we investigate whether the SP-EF data elements and the SOTM support measuring the influence of context factors and practice adherence on software outcomes.

Chapter 7

Quantifying Security Context Factors in Software Development

In this chapter, to address *RQ3: How do context factors and security practice adherence affect software development security outcomes?*, we present an evaluation of SOTM for quantifying security practice use and outcomes during software development. The four constructs (with abbreviated names for use in the modeling software) of SOTM are:

1. Software Development Context Factors (Development Risk) - measures of software characteristics that have been shown to be associated with vulnerabilities and defects;
2. Software Usage Context Factors (Usage Risk) - measures of software usage characteristics associated with the value an attacker will find in conducting a successful attack;
3. Practice Adherence (Adherence) - measures of the development team's security assurance efforts;
4. Security Outcomes (Outcomes) - measures of security-related indications (e.g. static analysis alerts, publicly reported vulnerabilities) associated with a piece of software over the course of the software's life cycle.

As described in Section 4.4, our constructs are rooted in the Common Criteria concepts as applied in the context of software development and use. To assess their utility in evaluating software development security, we hypothesize that the four constructs are related as follows:

- **H1** Usage Risk is associated with negative Security Outcomes
- **H2** Development Risk is associated with negative Security Outcomes

- **H3** Development Risk is inversely associated with Practice Adherence

We conduct case studies of the construct relationships, applying data from OpenHub ¹ and the National Vulnerability Database ² (NVD) to evaluate the model and test our hypotheses.

Our contribution is an empirical evaluation of the proposed model and metrics using two open source datasets.

The remainder of this chapter is organized as follows: Section 7.1 presents how we translate SOTM in to an analyzable SEM model. Section 7.2 presents our study methodology. Section 7.3 presents the case study and results. Section 7.3.7 discusses the measurement results. Section 7.4 presents our study limitations. Section 7.5 presents our conclusion.

7.1 Expressing SOTM as a SEM Model

To quantitatively analyze SOTM, we express it as a SEM model. We used the R ³ lavaan package to conduct our SEM analysis, as well as the ggplot2, and semPlot R packages. We now introduce the lavaan syntax for SEM models, and explain the semantics of each syntax element.

- Regression relationships between latent variables are specified using the $\sim$ operator (see Table 7.1). For example we translate hypotheses H1 ('Usage Risk is associated with negative Security Outcomes') and H2 ('Development Risk is associated with negative Security Outcomes') into the model as *Outcomes* $\sim$ *DevelopmentRisk* + *UsageRisk*. Establishing parameter estimates for these relationships allows us to test the hypotheses.
- Covariance relationships are specified using the $\sim\sim$ operator.
- Latent-measurement variable relationships are specified using the $=\sim$ operator, e.g.: *LatentVariable* $=\sim$ *MeasuredVariable1* +
- Dashed lines indicate estimates established by the researcher, or by the software. We have two examples of modeled fixed parameters in our structural model: We specify the absence of a direct relationship between Usage Risk and Development Risk (syntax: *SoftwareRisk* $\sim\sim 0 * \textit{UsageRisk}$), as we expect the constructs to be independent of each other. We specify the absence of a direct relationship between Adherence and Outcomes, as we expect Adherence to affect Outcomes through being moderated by overall

¹<https://www.openhub.net/>

²<https://nvd.nist.gov/>

³<https://www.r-project.org>

Development Risk. The remaining dashed lines are estimates fixed by the software, where it has estimated starting values in the course of solving the system of equations expressed by the model.

Table 7.1 SOTM Lavaan Syntax

Syntax	Symbol	Name	Description	Example
=~	$\bigcirc \rightarrow \square$	is measured by	=~ specifies how a latent variable (left side) is measured by the constituent variables listed on the right side	<i>DevelopmentRisk</i> =~ <i>SLOC</i> + <i>Churn</i>
~	$\bigcirc \leftarrow \bigcirc$	regression	~ specifies a regression of the dependent variable on the left-hand side to the independent variables on the right hand side of the expression.	<i>Outcomes</i> ~ <i>DevelopmentRisk</i> + <i>UsageRisk</i>
~~	$\bigcirc \leftrightarrow \bigcirc$ $\square \leftrightarrow \square$	undirected covariance	model a covariance relationship, but leave the direction of influence unspecified. When one side is multiplied by 0, the meaning is that the factors explicitly do not covary.	<i>UsageRisk</i> ~~ <i>Adherence</i> <i>Adherence</i> ~~ $0 * \textit{Outcomes}$

We now present the complete set of structural model constructs and relationships for SOTM

in the lavaan model syntax:

$$\begin{aligned}
&DevelopmentRisk = \sim DevelopmentRiskContextFactors \\
&UsageRisk = \sim UsageRiskContextFactors \\
&Outcomes = \sim OutcomeMeasures \\
&Adherence = \sim AdherenceMeasures \\
&Outcomes \sim DevelopmentRisk + UsageRisk \\
&DevelopmentRisk \sim Adherence \\
&UsageRisk \sim Adherence \\
&DevelopmentRisk \sim 0 * UsageRisk \\
&Adherence \sim 0 * Outcomes
\end{aligned} \tag{7.1}$$

7.2 Case Study Methodology

In this section, we present the steps required for analyzing a data set in terms of the SOTM.

7.2.1 Step 1: Select case study subject

Select, or prepare, a data set containing measurements of variables that can be related to the structural model's Usage Risk, Development Risk, Adherence, and Outcomes constructs. The unit of analysis is a software project at a point in time (e.g. OpenSSL as of April 2017), and measurements must be taken, or aggregated to the level of the project as of the data collection date. For example, a given value for SLOC is dependent on when the project is measured, and may need to be aggregated to the project level from a list of SLOCs for the component files of the project. Table 4.1 is a specification for a complete data set of measurement model metrics and how they relate to the constructs. Where complete SP-EF data is not available, collecting data representative of the measurement model metrics supports theoretical replication of the structural model.

7.2.2 Step 2: Link data source variables to the model constructs

Evaluate whether each dataset variable corresponds to one of the measurement variable definitions as described in Table 4.1. Where the dataset variable corresponds to a measurement variable, associate the dataset variable with the measurement variable's construct. For example,

if the dataset contains a code churn metric, associate the code churn metric with Development Risk.

7.2.3 Step 3: Evaluate collected data for fit problems

Data problems, for example noisy or non-normally distributed data, can cause model fit problems independent of the model’s quality. We excerpt Kline’s [37] advice on data preparation, focusing on the recommendations we applied in the course of our investigation.

- Normality: Standard SEM approaches assume multivariate normal relationships between variables in both structural and measurement models, requiring researchers to pre-examine the data for non-normal relationships. Calculating skewness and kurtosis for individual variables, and plotting combinations of variables can assist researchers in assessing the normality of their data. Potential solutions where non-normal data is found include excluding outliers, transformations such as log or square root, and use of estimation algorithms designed to accommodate non-normality.
- Collinearity: Collinearity between measurement variables affects model fit. We check for collinearity between measurement variables in the dataset, and drop collinear variables (as measured by a Spearman correlation coefficient greater than 0.7) that are theoretical alternatives.
- Outliers: Outliers are values that are very different from other values for the variable, where difference is measured by applying one of a set of heuristics for calculating difference and a decision rule for the border between outliers and typical values.
- Relative variances: As SEM solves for the covariance or correlation of variables with each other, SEM depends on the variances of the measured variables to be within an order of magnitude of each other, and typically in the range of 1–10. In this work, where a measured variable variance exceeds the variances of other measured variables by an order of magnitude or more, we create a transformed variable taking the log of the sum of the original variable value and a small constant, 1.

7.2.4 Step 4: Estimate SEM model

The specified relationships in the structural and measurement models represent a system of equations, as shown in Equation 7.1. Encode the combined structural and measurement models

in a SEM modeling tool, and run the tool to obtain estimates for the model (for more detail on SEM estimation, see Section 2).

7.2.5 Step 5: Test model fit

Once a set of estimates has been generated for a given model and dataset, SEM users evaluate fit measures and residuals to assess the suitability of the model. Model fit indicators and residuals both represent the degree of fit (or misfit) between the model and the dataset.

No single SEM fit measure captures all of the diagnostic information available, so SEM theorists and practitioners recommend reporting multiple goodness-of-fit and badness-of-fit measures. The model fit measures recommended by Kline [37] are as follows:

- Ratio of χ^2 to degrees of freedom. Report the calculated model χ^2 , its degrees of freedom and p-value. Ratios of 3 or less for χ^2 to degrees of freedom indicates acceptable fit.
- Steiger-Lind Root Mean Square Error of Approximation (RMSEA) - RMSEA is a ‘badness-of-fit’ measure, where values less than 0.10 indicate acceptable fit.
- Bentler Comparative Fit Index (CFI) - CFI compares fit of the researcher’s model to a baseline model, where values of 0.90 or greater indicate acceptable fit.
- Standardized Root Mean Square Residual (SRMR) - SRMR is a ‘badness-of-fit’ measure of the difference between the observed and predicted correlations. Zero is a perfect score, scores below 0.08 indicate acceptable fit.

7.2.6 Step 6: Perform Re-specification, if necessary

Kline [37] and Loehlin [47] both offer methodologies for diagnosing fit issues and revising the model in principled ways. We present a set of steps derived from these sources for application to our data and measurement model. We declare alterations to the structural model out of scope for the present work, as the work is intended to assess our structural model.

If model fit indicators show poor fit between the data and the model, it is common to consider adding, dropping, or moving measurement variables, if theory supports doing so. Modifying the model to achieve good fit is only good practice if justified by the theory underlying the model variables and relationships. In the present study, we do allow a list of transformations for the measurement model, as follows:

- Relative variances: SEM requires measurement model variable variances to be within a narrow range of each other, to avoid ill-scaled covariance matrices, supporting convergence

when performing model estimation. Transforming a variable by taking its log, square root, or multiplying by a constant is at the discretion of the researcher (transformation must be documented).

- Choice of measurement variable to construct association is at the discretion of the researcher.
- Where more than one measurement variable measures the same concept (e.g. team size measured by both a count and by an ordinal variable), variable choice is at the discretion of the researcher.

If transforming the data does not yield adequate fit, the next step is to evaluate the measurement model. Loehlin [46] recommends, as a first step for respecification, fitting the data to a model in which the latent variables are completely intercorrelated, yielding a perfectly fitting structural model, revealing fit difficulties rooted in the measurement model.

In addition to the global fit indicators presented above, researchers must examine residuals to assess local model fit. Residuals, also called error terms, are associated with each measurement variable, and represent variance not explained by the construct with which the measurement is associated. Per Kline [37], p. 278, residual values near zero indicate that the construct accounts for the bulk of the variance in the measurement variable. Residual values greater than 0.1 indicate that the construct does not account for the bulk of the variance in the measurement variable, and should prompt investigation, re-specification, and/or explanation. Residual values provide a diagnostic of model fit.

7.2.7 Step 7: Report Results

Kline [37] recommends reporting model fit in the terms of the global fit indicators, and in terms of comparison between the expected theoretical relationships embedded in the model and the actual parameter magnitudes and signs observed in the data. In this work, we apply basic interpretations, focusing only on the sign and magnitude of each parameter estimate, as compared to our theorized expectations, where sign indicates direction of influence, and magnitude indicates effect size. For the parameter estimates of measurement variables associated with a single latent variable, sign indicates direction of influence, and (standardized ⁴) parameter estimates indicate the relative importance of each measurement variable's effect on the latent variable.

⁴standardized as a correlation, computed as the variable pair's covariance divided by the square root of the product of the variable pair's variances

For latent variable relationships, sign indicates the direction of influence, and magnitude indicate the relative importance of the latent variable’s effect on the receiving latent variable, as compared with the magnitude of the other latent variable parameter estimates.

7.3 Case Study

This section presents a case study of the structural and measurement models using existing software development security data.

We presented the data elements to be collected for our full model in Section 4, and the data collection guidebook [62] for the measurement model gives instructions on how to collect the data for a software development project. SEM is a large-sample technique, with median sample size in the literature of 200 observations [37]. The need for large quantities of software development security data leads us to examine existing software development security datasets. Further, confirmation of our hypothesized structural and measurement relationships in data we did not generate strengthens the case for the theorized relationships.

At present, no single dataset contains a set of 200+ projects with complete data collected using the SP-EF framework. However, we have identified two datasets that contain most of the information required for measurement model for the Development Risk, Usage Risk, and Outcomes constructs:

- Black Duck Software ⁵ maintains OpenHub ⁶, a tracking site for open source software projects. Based on the OpenHub database, Nagappan et al. [69] built and published a dataset of 20,028 projects (OpenHub) to enable assessment of diversity in software engineering research. The OpenHub dataset contains fields related to our Development Risk and Usage Risk constructs, as described below in Section 7.3.1.
- The U.S. National Institute of Standards and Technology (NIST) maintains the National Vulnerability Database (NVD) ⁷, an online database of publicly reported software vulnerabilities, with over 79,000 vulnerabilities dating back to 1988. Vulnerability reporters assign each vulnerability a Common Vulnerability Scoring System (CVSS) score and associated CVSS base metrics, according to the scheme defined in the CVSS guide [55].

By combining these datasets, we can establish a baseline for assessing the SOTM constructs and their relationships. Our unit of analysis is the software development project. Each Openhub

⁵<https://www.blackducksoftware.com/>

⁶<https://www.openhub.net/>

⁷<https://nvd.nist.gov/>

record contains a set of descriptive data about a single project. Each NVD record contains a reported vulnerability for a project. We summarize NVD vulnerability counts for each project. We use the OpenHub projects as our baseline, and include vulnerability counts for OpenHub projects where they have matches in the NVD dataset. The absence of a vulnerability record in the NVD does not mean that a project has no security vulnerabilities, only that vulnerabilities have not been reported. Projects may have no reported vulnerabilities for a variety of reasons, for example, because they are not paid attention to by researchers and attackers, or because they are relatively vulnerability-free. To limit the effect of unreported vulnerabilities on the results, we restrict our analysis to those projects which have NVD vulnerability records.

7.3.1 Data selection

In this section, we describe how we selected each observation and each variable in the source datasets, and how we treated and merged the data. We first present how we interpreted each source dataset field in terms of our structural model concepts. We then present how we merged the datasets into a combined dataset.

The project record contains descriptive data, e.g. project name and version, and security-relevant metrics, e.g. total lines of code (SLOC), and contributor count (Team Size). We now present our mapping of the OpenHub and NVD fields to our SOTM measurement model metrics.

7.3.1.1 Usage Risk

We map the OpenHub `user_count` metric to our Number of Users metric. Number of Users is one of multiple metrics associated with Usage Risk (Table 4.1), so this model is a partial account of the factors affecting Usage Risk.

7.3.1.2 Development Risk

We model the following Development Risk measurement model metrics based on OpenHub data fields:

- SLOC - `total_code_lines` (`total_code_lines`)
- Team Size - `twelve_month_contributor_count` (`contributor_count`)
- Product Age - difference in months between `min_month` and `max_month` (`product_age`)
- Churn - `code_churn_12months` (`code_churn`)

7.3.1.3 Adherence

We do not have direct measures of security practice adherence available in our datasets. We evaluate a version of SOTM without the Adherence construct, to study the relationships between the remaining three constructs as reflected in the available data.

7.3.1.4 Outcomes

We obtain a metric for Outcomes by counting per-project vulnerabilities as of the end of 2012 for each project in the NVD. We treat each unique software name in the NVD records as a distinct project, and sum all vulnerabilities for a project, reflecting our measurement model vulnerability count metric.

7.3.2 Data collection

For each project, Openhub included SLOC, Language, Contributor Count, Code Churn over the preceding 12 months, Commits over the preceding 12 months, Project Age, and Project Activity. Nagappan et al's [69] inclusion criteria required that each project had at least two committers between June 2011 and June 2012, complete data for all collected fields, and no invalid data (e.g. negative SLOC.)

We included all 20,028 OpenHub projects in the 'Combined' dataset ⁸. We present summary statistics for the OpenHub dataset in Table 7.2. We grouped NVD records by project name, and summed vulnerability counts as of the end of 2012 for each project name (in keeping with the time period represented by the OpenHub dataset). We then augmented the Openhub project data with a vulnerability count field, reporting the NVD vulnerability count for the 698 projects that matched by name, and 0 for the remaining projects. We dropped one project, DD-WRT ⁹, which had a `total_code_lines` value of 258 million lines of code, roughly triple the size of Windows, as an outlier, yielding a dataset of 697 projects for further analysis.

7.3.3 Estimation

Combining the structural and measurement models we have defined with the subset of data available in the Combined dataset, we have the Simplified Model definition, expressed in lavaan syntax:

⁸We make our scripts for all described steps available online at www.github.com/pjmorris/Quantifying

⁹www.ddd-wrt.com

Table 7.2 Combined Demographics for 697 Projects

Statistic	Mean	St. Dev.	Min	Max
total_code_lines	666,297.7	2,18,927.0	56	26,915,903
twelve_month_contributor_count	34.6	90.1	2	1,167
project_age	99.2	55.70	2.0	355.2
code_churn_12months	544,547.2	2,195,992.0	0	25,239,730
CVECount	11.3	45.8	1	776
DevAttention	0.04	0.7	0.0	18.0
user_count	260.7	901.4	0	11,150

$$\begin{aligned}
DevelopmentRisk &= \sim total_code_lines + twelve_month_contributor_count + \\
&\quad project_age + code_churn_12months \\
Outcomes &= \sim CVECount \\
Outcomes &\sim DevelopmentRisk + UsageRisk \\
UsageRisk &= \sim user_count
\end{aligned} \tag{7.2}$$

7.3.4 Model Fit

To avoid estimation problems caused by ill-scaled covariance matrices (high ratio between the largest and smallest variances), Kline [37] recommends rescaling variables with low or high variances relative to the other variables in the dataset. We implemented rescaling by applying R's scale function defaults to each variable, subtracting the column mean from each variable, and dividing each variable by its standard deviation.

Standard SEM estimation assumes multivariate normally-distributed variables [37], pp. 74-78, and normally-distributed joint distributions between variables, however SEM methodologists have developed procedures and estimators for non-normal data. As our data consists primarily of counts, we checked for skewness (ranging from 2.29 for user_count to 76.48 for total_code_lines) and kurtosis (ranging from 12.23 for project_age to 7993.58 for total_code_lines), indicating that we have varying degrees of non-normality in our data. Where data are expected to be non-normal, as with counts, Kline [37], pp.238-9, recommends using robust maximum likelihood (RML) to estimate the model, as it does not assume normality, but estimates parameters for each variable's distribution based on the data for each variable. Lavaan implements RML

Table 7.3 Global Fit Measures and Results

Fit Measure	Threshold	Simplified	Respecified
Number of observations		697	697
Model chi-square		36.35	41.41
Model d.f.		8	6
Model p-value	$\leq$ 0.01	0.0	0.0
Robust RMSEA	$\leq$ 0.10	0.15	0.12
Robust CFI	$>$ 0.90	0.88	0.94
SRMR	$<$ 0.08	0.07	0.048

through the MLR estimator.

Applying the described estimation procedure to our transformed data ¹⁰, yielded global fit indicators that were outside the range of standard fit criteria thresholds for CFI, RMSEA and SRMR, as shown in the Respecified column of Table 7.3. We examine fit refinement in the next section.

7.3.5 Re-specification

After reviewing the lavaan modification index recommendations and what we expect the data to mean, we added co-variance relationships, as follows:

- *total_code_lines* $\sim\sim$ *code_churn_12months*. We reason that the relationship is reasonable in light of theory because larger projects tend to have more lines of code available to change.
- *twelve_month_contributor_coin* $\sim\sim$ *code_churn_12months*. We reason that the relationship is reasonable in light of theory because larger numbers of contributors are likely to change more code.

¹⁰As a reminder, our scripts are available at https://github.com/pjmorris/paper_modeling-sp/blob/master/CombinedCaseStudy.R

Table 7.4 OpenHub-NVD Respecified Model Results

<i>Latent Variables:</i> ~ Measured variables	Estimate	Std.Err	z-value	$P(> z)$
<i>DevelopmentRisk</i> =~				
total_code_lines	0.459			
project_age	0.11	0.25	0.99	0.33
code_churn	0.77	0.79	2.14	0.03
contributor_count	0.88	0.57	3.35	0.01
<i>UsageRisk</i> =~				
user_count	1.000			
<i>Outcomes</i> =~				
CVECount	1.000			
<i>Adherence</i> =~				
DevAttention	0.24			
Regressions:				
<i>Outcomes</i> ~				
DevelopmentRisk	0.40	0.48	1.81	0.07
UsageRisk	0.34	0.19	1.84	0.07
<i>DevelopmentRisk</i> ~				

The re-specified model had global fit characteristics within the traditional fit criteria thresholds, as shown in the Respecified column of Table 7.3. We present the parameter estimates for the Respecified model in the results, and discuss the implications in Section 7.3.7.

7.3.6 Reporting Results

We present the global fit results for the Simplified and the Respecified models in Table 7.3. We report the estimated parameter values, standardized, for the Respecified structural and measurement models in Table 7.4 ¹¹. We present the standardized parameter estimates, and the residuals, in the context of the full structural and measurement models in Figure 7.1.

Interpreting the (standardized) parameter estimates in terms of our hypothesized construct relationships, we have the following:

- Usage Risk is correlated (0.34) with Security Outcomes, but the relationship is not sta-

¹¹Standardized SEM parameter values are correlations, and can be interpreted as for regression.

Figure 7.1 Respecified OpenHub-NVD Combined Model

tistically significant (p-value = 0.07).

- Development Risk is correlated with (0.40) with Security Outcomes, but the relationship is not statistically significant (p-value = 0.07)

The signs and magnitudes of the parameter estimates are as theorized for each of the hypotheses, however the relationships are not statistically significant for the model, data, and estimator used.

The residual variance values, shown in Table 7.5, are lower than the .10 guideline established in Kline [37], with the exception of total_code_line’s relationship with project_age, and user_count’s relationship with project_age.

Table 7.5 OpenHub-NVD Respecified Model Residuals

	1.	2.	3.	4.	5.	6.
1. total_code_lines	0	0.11	0	0	-0.05	0.02
2. project_age	0.11	0	-0.03	0.05	0.04	0.21
3. code_churn_12months	0	-0.03	0	0	-0.01	0.01
4. twelve_month_contributor_count	0	0.05	0	0	-0.01	-0.01
5. CVECount	-0.05	0.04	-0.01	-0.01	0	0
6. user_count	0.02	0.21	0.01	-0.01	0	0

Returning to the research question, ‘*RQ3* How do context factors and security practice adherence affect software development security outcomes?’, we found the theorized relationships for the hypothesized relationships of Development Risk and Usage Risk with Security Outcomes, however the relationships are not statistically significant in our data. We must collect further data to test the hypothesized relationship of Practice Adherence with Development Risk.

7.3.7 Discussion

In this section, we first compare our expectations with our findings for our case study results. We then present implications from our findings for our method, model, and data.

In our data, vulnerability counts are as influenced by the number of people using the software as they are by traditional software development risk metrics. We found effects for both Usage Risk and Development Risk on Security Outcomes, with Usage Risk having an effect (0.34 standardized) comparable to that of Development Risk (0.40 standardized). As a check

on this finding, we ran two linear regressions on CVECount, using a) all the other variables, and b) all the other variables without user_count. Adjusted R^2 without user_count was 0.26, Adjusted R^2 with user_count was 0.37. Our data and analysis suggest that usage is a significant factor associated with publicly reported vulnerabilities, as predicted by, e.g., Zimmermann et al. [105]. Software usage must be taken in to consideration when evaluating software security. In our data, the Development Risk metrics most influential on vulnerability counts, by rank, were number of developers, code churn, and SLOC. Development Risk is correlated most strongly with twelve_month_contributor_count (0.88), followed by code_churn (0.77), total_code_lines (0.46), and project_age (0.11, not statistically significant, p-value=0.33). Given the single measurements of each construct, we have no information on the relative importance of the measurements for the Usage Risk, Adherence, and Outcomes constructs.

Our measurement model metrics are correlated with vulnerability counts. The datasets we studied did not contain a complete set of the measurement model metrics we are interested in, as listed in Table 4.1. For the metrics the datasets did contain, we found statistically significant support for each of them, with the exception of project_age.

Our case study model and metrics only partially explain software development project vulnerability counts. We expect that at least one reason for the small combined effects of Usage Risk and Development Risk on Outcomes (Respecified SEM model, R^2 0.38) is due to the underlying measurement variables being incomplete accounts of the constructs they measure. For example, only the number of users is considered for Usage Risk, not the theorized effects of, e.g. number of machines, number of dollars affected, or the presence of sensitive data. Similarly, the Development Risk measurements do not include, at present, language, operating system, domain, or the other variables that have been identified as contributing to security issues in software. Measuring the R^2 of our constructs and measurements correlation with Outcomes gives us an assessment of the efficacy of our model and framework, a means for measuring the effect of changes to the structural and measurement models.

Stepping back from the details of the case study measurements, we propose three benefits of the model we have presented and evaluated:

- By providing parameter estimates and p-values for their relationships, the structural model constructs provide quantitative guidance for the value and predictive power of measurements collected when evaluating software security.
- SEM fit indicators and parameter estimates provide numerical assessment of model and metric performance, enabling data-driven assessment and iterative improvement of metric use.

- Because the structural model does not depend on a particular set of measurements, it can be applied at other granularities than the software development project example used in this paper. In future work, we intend to apply the model to evaluate binaries, source files, and commits.

We have built a model, identified metrics to collect, collected data, and analyzed the data, supporting the notion that Usage Risk and Development Risk both correlate with security Outcomes. Further development of the structural model and its constructs, and the measurement model metrics and their measurement and collection should provide further insight into the software development and usage constructs affecting security outcomes for the software’s developers, managers, and users.

7.4 Limitations

We now discuss threats to validity of our study.

Our two datasets represent thousands of open source and commercial software projects. However, each dataset represents a restricted subset of software development projects, where the NVD dataset is constrained to projects with CVE records, and the OpenHub dataset is constrained to open source projects as chosen by the site’s administrators. Our results are constrained to open source projects reported on by OpenHub that also have vulnerabilities reported in the NVD. Generalizing to proprietary projects, and to projects that have security vulnerabilities reported by other means, and to projects that do not have vulnerabilities will require alternate data sources.

Kaminsky [34] critiqued the NVD data, pointing out that the existence of a vulnerability record is more indicative of reporter and finder interest in the software than of the software’s quality. The strength of the effect of `user_count` on Outcome shown in our analysis offers empirical evidence for Kaminsky’s concern. We view reporter and finder interest as indicative of the kind of usage risk we seek to measure, distinct from software quality. Further work comparing software quality between samples of non-NVD projects and NVD projects is needed to establish the strength of the effect of reporter and finder interest, and its effect on usage risk.

Use of the NVD vulnerability counts is a limitation, as they are externally reported and may understate the presence of security issues. Where software development teams track vulnerabilities and related security issues internally, that data could be used to increase the model’s accuracy.

The variety of factors involved in security measurement suggest that further investigation

is necessary. Complete validation of the model would require use of a variety of frameworks, metrics, and data sources to evaluate the constructs and their relationships. That said, we used two independent data sources, increasing confidence in the theorized signs and magnitudes of the correlations found in the data sets, mixed with caution due to the lack of statistical significance of the relationships.

In terms of construct validity, we propose a structural model of factors we believe to be relevant, and a measurement model based on the literature, but we leave room for augmenting the existing set of factors and the measurements taken on those factors. The analytical tools of SEM provide diagnostics to check for residual error and modification potential, enabling iteration over the structural and measurement models to account for additional factors in the model.

The two datasets we used each contain subsets of the variables we theorize are necessary to assess security posture. We expect that the missing variables influence both the relative measures of each factor, and of the relationships between each factor.

In particular, we acknowledge the absence of Adherence in the version of the model evaluated in the case study. We have begun developing adherence measures (e.g. in Morrison [67]), and intend to evaluate and incorporate these adherence measures in future work.

Statistical model-building in software engineering often uses Bayesian Belief Networks rather than SEM, e.g. Fenton [19]. Judea Pearl has claimed the two techniques are essentially identical, preferring SEM when the research question is of the form ‘What factors determine the value of this variable?’ - ¹² We view our task in terms of determining the factors behind the values of the modeled variables, leading us to cast the model in terms of SEM.

7.5 Contributions of the Quantification Study

In this chapter, we have presented a model of factors affecting software security, with empirical tests of the model using two datasets. Our results indicate:

- In the OpenHub-NVD data, Usage Risk, as measured by user count, has a comparable correlation with Outcomes to Development Risk, as measured by SLOC, Churn, Contributor Count, and project age. Factors outside the team’s direct influence have to be considered when evaluating security performance and mitigations

¹²<http://causality.cs.ucla.edu/blog/index.php/2012/12/07/on-structural-equations-versus-causal-bayes-networks/>

- Our data corroborate previous research findings that team size, code size, and code churn are correlated with discovered vulnerabilities, and do so while controlling for other factors influencing security Outcomes. Measuring the relative impact of each measurement on its construct, and on the model's performance as a whole, supports refining the measurement framework and the theoretical model as further data are collected and evaluated.
- Stepping back from the specifics of the case studies, SEM offers means of assessing the relative importance of the measurements taken for software security assessment.

Our data suggest that not only software attributes, but the context of software use must be accounted for to assess the state of software security. Researchers and practitioners should measure both software attributes and software usage context when assessing software development practice adherence. That said, our analysis shows correlation, but not causation. Further work including manipulation of variables must be conducted to assess the causes of software insecurity and security.

Chapter 8

Case Studies using SP-EF

In this chapter, we apply SP-EF to conduct case studies of software practice adherence over time in four open source software development projects.

We focus on two research questions in the case studies:

- **RQ2.1:** Can the complete set of SP-EF measurements be collected for a software development project?
- **RQ3:** How does security practice adherence affect software development security outcomes?

The remainder of the chapter is organized as follows: Section 8.1 presents an overview of how SP-EF is applied to measure a software project, Section 8.4 presents a case study of phpMyAdmin, Section 8.7 presents a discussion of the case study results, Section 8.8 reports limitations, and Section 8.9 concludes.

8.1 Methodology

While SP-EF studies will vary in choice of subjects, goals, and research questions, data collection should collect the elements described in Appendix C.1, adapted only to reflect how the project data is kept by the project team, and how it is shared with the researchers. One approach to using SP-EF is to talk with the project staff and read through the project documentation for objective and subjective evidence of the security practices. Writing down actual values where possible, or estimates, for the Context Factors, objective and/or subjective practice adherence measures, and outcome measures may be suitable for an informal study of a small team. A development team could evaluate itself, and use the evaluation to guide discussions

about the teams security posture. Where the source code and its history of changes is stored in a version control system, Language(s), SLOC and Churn, and Developers can be calculated, and the source code and change history can be text-mined for objective measures of practice adherence. Commit comments sometimes contain pertinent summaries of reasons for changes, as well as references to defects resolved by the changes. Researchers should collect the objective practice adherence metrics from at least the project documentation, the project issue tracker and developer email lists. Treat each issue and email as an individual item to be classified. Where possible, the subjective practice adherence measures should be also collected, through interviews and/or surveys. Version control systems record what the team does to source code, and issue trackers record what the team plans to do. Email archives offer a view of what the team talks about. Researchers should obtain the objective practice adherence measures by identifying their mentions in the issue tracker issues and the developer email list.

8.1.1 Subject selection

Given our goals of collecting SP-EF measurements and studying whether security practice adherence affects security outcomes, we select projects based on availability of the following data:

- Records of software security vulnerabilities.
- Version control system access, providing both project source code and the history of changes to the code over time.
- Bug tracker system access, providing records of vulnerability and defect discovery and resolution.
- Project documentation, providing access to information about the projects development process and practices.
- Survey responses from the survey described in Chapter 6.

We focus on four of the eight projects that met the above criteria: BitCoin, IBM wasCloud (studied in Section 5), phpMyAdmin and OpenSSL.

8.1.2 Data Collection

We collect project documentation and history using the projects website, version control system, and bug tracker, as primary sources, and as sources for links to further information. For the

selected projects, we collect the project’s NVD CVE records, the developer email list archive, the commit message history, and the defect tracking system messages. In addition, we reviewed each project’s website. We use the term ‘source’ to refer to the type of message, e.g. email, commit, issue.

Data collection steps:

- We record and report source code and development team size for each month using version control data.
- We record and report vulnerability and defect counts for each month using bug tracker data.
- We record and report practice use occurrences for each month using the practice use data we collected. For each practice, we repeated the procedure described in Section 5.1 for each project: I read and classified the set of issue tracking records from each project according to the guidelines for identifying the presence of SP-EF practices.
- One graduate student, and one undergraduate student received training in SP-EF, classified a randomly selected pool of issues. We compared their results to the classifications I generated.

8.1.3 Research question analysis

For this study, our measure of practice adherence is its presence, as measured by the earliest reference to the practice in the project issues we classified. We record the practice as starting at the time of the earliest occurrence in our sample of issues from each project. For this study, our measure of practice adherence is a count of security practice events, where each event represents one or more references to a security practice in a bug tracking issue, email, or commit message. To study how security practice adherence relates to security outcomes over time, we adopted Project Month as our measurement time period, aligning all measurements collected to the calendar month.

To evaluate RQ2.1, we compare the full list of SP-EF measurements (Table 4.1) with the data collected from each of the four case study projects.

To evaluate RQ3, we track two outcome measures, Vulnerability Density (Vdensity) and Vulnerability Removal Effectiveness (VRE). VDensity [1] is the of number of discovered vulnerabilities per 1000 SLOC. Lower values for Vdensity may indicate high code quality and/or opportunities for discovering latent vulnerabilities. VRE is the ratio of pre-release vulnerabilities to

total vulnerabilities found, pre- and post-release, analogous to defect removal effectiveness [35]. Higher values for VRE indicate the development process effectiveness at finding vulnerabilities. We examine the relationship between our practice adherence metrics and Vdensity and VRE.

8.2 Bitcoin Case Study

Bitcoin is an experimental digital currency that enables instant payments to anyone, anywhere in the world ¹. Bitcoin uses peer-to-peer technology to operate with no central authority: managing transactions and issuing money are carried out collectively by the network. Nakamoto's [70] claim for Bitcoin security is that the system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes, a concept known as 'Proof of Work'. In the eight years of its existence, Bitcoin has attracted billions of dollars of investment ², highlighting the need for security.

Bitcoin provides an open defect repository and version control system, both based on a public Github repository ³. The Bitcoin case offers an opportunity for us to investigate whether SP-EFs practice adherence measures will show evidence of security practice use where we expect the development team to pay attention to security.

8.2.1 Data Collection

We cloned the Bitcoin github repo, representing the source code and changes made during the history of the project. We applied CVSAAnaly from the MetricsGrimoire tool set reported on by Gonzalez-Barahona [24] to process and summarize the github data, and then analyzed and reported on the data using R ⁴. We extracted email data from downloaded developer email archives. We extracted defect and vulnerability data from Bitcoins github repository, and from the projects CVE records ⁵.

For each artifact we identified, we recorded the document name, URL, age, and made note of any available change history, e.g. wiki page change records. We manually classified pre- and post-release vulnerabilities based on a study of who reported the vulnerability and whether they were identifiable as a Bitcoin developer (by whether the email address was associated

¹<https://github.com/bitcoin/bitcoin>

²<https://blockchain.info/charts/market-cap?timespan=all>

³<https://github.com/bitcoin/bitcoin>

⁴<https://www.r-project.org/>

⁵https://www.cvedetails.com/vulnerability-list/vendor_id-12094/Bitcoin.html

with commits, as well as through examination of the project’s records of how their CVEs were discovered and resolved ⁶).

8.2.2 Results

In this section, we present the SP-EF measurements for Bitcoin and the research question results. We present a summary of the SP-EF measurements for Bitcoin in Table 8.3.

8.2.2.1 Context Factors

Bitcoin’s Domain is online payments, and the languages it is written in are C, and C++. The context factors Project Age, SLOC, Churn, Developers, and User Count evolved over time as shown in 8.1a. SLOC, Churn, and Developers are based on version control system data, while User Count is modeled using the number of Bitcoin addresses ⁷. We, subjectively, rate Bitcoin’s Confidentiality Requirement and Integrity Requirement, and Availability Requirement as High, because the software manages ‘worldwide’, ‘instant’ financial transactions.

8.2.2.2 Practice Adherence

In this section, we review security practice usage results obtained from researcher assessment of Bitcoin. We present observations organized by security practice, and include links to evidence, where available, in Table 8.1. Figure 8.1b presents the first occurrence of each practice in the timeframe presented, as measured by the issues we classified. Italicized quotes are from Bitcoin project communications.

- Apply Data Classification Scheme. We were not able to identify documented process or artifacts for data classification.
- Apply Security Requirements. Bitcoin’s security requirements can be traced back to Nakamoto’s [70] proposal for the project, where cryptographic proof is proposed as an alternative to trust for conducting online financial transactions.
- Apply Threat Modeling. The developers discuss changes in terms of a security model, though the model is not contained in a document we have yet identified. *If that sole property is desirable, then sure, add it. But it isn’t reflective of the existing security model.*

⁶https://en.bitcoin.it/wiki/Common_Vulnerabilities_and_Exposures

⁷<https://blockchain.info/charts/n-unique-addresses?timespan=all>

Table 8.1 Bitcoin Manual Review Evidence

Security Practice	Source (Link)	Event Date
Apply Data Classification Scheme		8/2017
Apply Security Requirements	Nakamoto [70]	2008
Apply Threat Modeling	No explicit document, but developers refer to a ‘security model’	2/2011
Document Technical Stack	Dependencies https://github.com/bitcoin/bitcoin/blob/master/doc/dependencies.md	5/2011
Apply Secure Coding Standards	Contribution Standards https://github.com/bitcoin/bitcoin/blob/master/CONTRIBUTING.md https://github.com/bitcoin/bitcoin/blob/master/doc/developer-notes.md	12/2014
Apply Security Tooling	Fuzzing: https://github.com/bitcoin/bitcoin/blob/master/doc/fuzzing.md	2/2011
Perform Security Testing	Unit Tests https://github.com/bitcoin/bitcoin/blob/master/src/test/README.md Integration Tests https://github.com/bitcoin/bitcoin/tree/master/test	12/2011
Perform Penetration Testing		11/2011
Perform Security Review		10/2015
Publish Operations Guide	Developer Documentation: https://github.com/bitcoin/bitcoin/tree/master/doc User Documentation: https://en.bitcoin.it/wiki/Main_Page	
Track Vulnerabilities	https://bitcoin.org/en/bitcoin-core/contribute/issues#disclosure	4/2012
Improve Development Process		2/2011
Perform Security Training		

- Document Technical Stack. The developers maintain a dependencies document describing the components of Bitcoin, and the stack can be further characterized through the contents of the project make and build files. The project uses a deterministic build process ⁸ to assure that the expected source code and dependencies are included in the generated executables. The developers discuss updates to the stack via email and issue tracking. This patch enables several new GCC compiler hardening options that allows us to increase the security of our binaries
- Apply Secure Coding Standards The developers maintain a document describing how to contribute to the project, and a coding standards document.
- Apply Security Tooling. The developers fuzz test Bitcoin Core. We were not able to identify documented process for security tooling, however we found multiple mentions of Valgrind, a memory management analyzer, in the issues, and in the test suite.
- Perform Security Testing. The developers maintain automated unit testing and integration testing test suites.
- Perform Penetration Testing. We were not able to identify documented process or artifacts for penetration testing.
- Perform Security Review. The developers manage changes to the codebase through Github Pull Requests ⁹. The contribution guidelines call for each pull request to be peer reviewed.
- Publish Operations Guide. The development team maintains documentation for developers and users of Bitcoin, including security-related material.
- Track Vulnerabilities. The developers maintain separate vulnerability reporting processes for responsible disclosure, as well as a list of vulnerabilities resolved. The team tracks and resolves vulnerabilities as they are reported.
- Improve Development Process. The developers have refined the development process over time, for example by introducing fuzzing in December 2016.
- Perform Security Training. We were not able to identify documented process or artifacts for Security Training.

⁸<https://github.com/bitcoin-core/docs/blob/master/gitian-building.md>

⁹<https://help.github.com/articles/about-pull-requests/>

We present a display of the first occurrence of each practice, drawn from the sample of classified issues in Figure 8.1b.

8.2.2.3 Outcome Measures

We present Bitcoin CVE count per month (CVE), and as a running total (CVer_t), as well as Vdensity and VRE for 2010-2016 in Figure 8.1c.

8.3 IBM wasCloud Case Study

We have described the wasCloud project in Chapter 5, including the data collection and case study results. The wasCloud case offers an opportunity for us to investigate whether SP-EFs practice adherence measures show similar characteristics when measured through survey, text mining, and researcher observation in a context where we have direct access to the development team and its records. We present a summary of the SP-EF measurements for wasCloud in Table 8.3. Following, we present the longitudinal SP-EF measurements for wasCloud.

8.3.0.1 Context Factors

The context factors SLOC, Churn, Developers, and User Count evolved over time as shown in Figure 8.2a.

8.3.0.2 Practice Adherence

The project’s practice adherence has been qualitatively described in Section 5.2.1.2. In Figure 8.2b presents the date of the first occurrence of each practice, measured by the earliest date of an RTC issue classified for that practice.

8.3.0.3 Outcome Measures

We present Bitcoin CVE count per month (CVE), and as a running total (CVer_t), as well as Vdensity and VRE for 2013-2015 in Figure 8.2c.

(a) Bitcoin Context Factors

(b) Bitcoin Practice Adherence Keyword Counts

(c) Bitcoin Practice Adherence Outcome Measures

Figure 8.1 Bitcoin Metrics 2010-2016

(a) wasCloud Context Factors

(b) wasCloud Practice Adherence Keyword Counts

(c) wasCloud Practice Adherence Outcome Measures

Figure 8.2 wasCloud Metrics 20103-2015

8.4 phpMyAdmin Case Study

phpMyAdmin is a popular MySQL administration tool, with rich documentation including various books ¹⁰. phpMyAdmin provides an open defect repository based on and version control system, both based on a public Github repository ¹¹, and the project issues security advisories and releases for critical security issues. phpMyAdmin has participated in the Google Summer of Code9 annually since 2008 ¹². phpMyAdmins administrators have introduced documentation and process changes in conjunction with its participation ¹³. The phpMyAdmin case offers an opportunity for us to investigate whether SP-EFs practice adherence measures will show evidence of security practice use where we know in advance that such changes exist.

8.4.1 Data Collection

We cloned the phpmyadmin github repo, representing the source code and changes made during the history of the project, encompassing 2005 source files, and 99,121 commits by 955 unique developers. We applied CVSSAnaly from the MetricsGrimoire tool set reported on by Gonzalez-Barahona [24] to process and summarize the github data, and then analyzed and reported on the data using R ¹⁴. We extracted email data from downloaded developer email archives. We extracted defect and vulnerability data from phpMyAdmins github repository ¹⁵, and from the security section of its website [<https://www.phpmyadmin.net/security/>], and checked it against the projects CVE records ¹⁶.

For each artifact we identified, we recorded the document name, URL, age, and made note of any available change history, e.g. wiki page change records. We manually classified pre- and post-release vulnerabilities based on a study of who reported the vulnerability and whether they were identifiable as a phpMyAdmin developer (by whether the email address was associated with commits).

¹⁰<https://www.phpmyadmin.net/15-years/>

¹¹<https://github.com/phpmyadmin/phpmyadmin>

¹²<https://developers.google.com/open-source/gsoc/>

¹³e.g. <https://www.phpmyadmin.net/news/2008/4/4/google-summerof-code-2008-and-phpmyadmin/>

¹⁴<https://www.r-project.org/>

¹⁵<https://github.com/phpmyadmin/phpmyadmin/issues>

¹⁶<https://www.cvedetails.com/vendor/784/Phpmyadmin.html>

8.4.2 Results

We present a summary of the SP-EF measurements for wasCloud in Table 8.3. In this section, we present details of the SP-EF measurements for phpMyAdmin.

8.4.2.1 Context Factors

phpMyAdmins Domain is administrative tools, in particular, web-based database administration, and the languages it is written in are PHP, SQL, Javascript, and HTML. The context factors SLOC, Churn, Developers, and User Count evolved over time as shown in Figure 8.3a. SLOC, Churn, and Developers are based on version control system data, while User Count is estimated based on 20% of 200,000 monthly downloads reported in September 2013¹⁷, projected linearly from the project start date. We, subjectively, rate phpMyAdmins Confidentiality Requirement and Integrity Requirement as High, because the software supports administrator creation, editing, and deletion of MySQL database schemas and data. We rate phpMyAdmins Availability Requirement as Low, because the software is a optional, graphical alternative to command line utilities, and is not essential to MySQL database administration.

8.4.2.2 Practice Adherence

In this section, we review security practice usage results obtained from researcher assessment of phpMyAdmin. We present observations organized by security practice, and include links to evidence, where available, in Table 8.2. Figure 8.3b presents the first occurrence of each practice in the timeframe presented, as measured by the issues we classified. Italicized quotes are from phpMyAdmin project communications.

- Apply Data Classification Scheme. We were not able to identify documented process or artifacts for data classification.
- Apply Security Requirements. The team considers security on a case-by-case basis as code is written. The two main administrators have been with the project since shortly after its inception, and they monitor features, and issues for security as well as other quality attributes.
- Apply Threat Modeling. We were not able to identify documented process or artifacts around threat modeling.

¹⁷<https://www.phpmyadmin.net/15-years/>

Table 8.2 phpMyAdmin Manual Review Evidence

Security Practice	Source (Link)	Event Date
Apply Data Classification Scheme		
Apply Security Requirements	http://sourceforge.net/p/phpmyadmin/feature-requests/1499/	??
Apply Threat Modeling		
Document Technical Stack	Developers page: https://www.phpmyadmin.net/develop/ Developers wiki: https://wiki.phpmyadmin.net/pma/Development	Feb 2011
Apply Secure Coding Standards	https://wiki.phpmyadmin.net/wiki/index.php?title=Developer_guidelines&diff=6214&oldid=6213 PEAR Coding Standards - http://pear.php.net/manual/en/standards.php https://wiki.phpmyadmin.net/pma/Security_pitfalls	Aug 2010 Mar 2012
Apply Security Tooling	PHP_CodeSniffer Coveralls: code coverage Scrutinizer	Aug 2011 Sept 2013 May 2014
Perform Security Testing	https://github.com/phpmyadmin/phpmyadmin/commits/a7554b3d962f5df280828085bdb648a5321ee9d6/test?page=76 PHPUnit	Nov 2005
Perform Penetration Testing	https://wiki.phpmyadmin.net/pma/Unit_Testing Email exchanges with phpMyAdmin administrator confirms the project works with external security researchers to test new releases.	July 2009 ??
Perform Security Review		
Publish Operations Guide	http://docs.phpmyadmin.net/en/latest/	Apr 2001
Track Vulnerabilities	https://www.phpmyadmin.net/security	Jun 2003
Improve Development Process	https://wiki.phpmyadmin.net/pma/Developer_guidelines https://wiki.phpmyadmin.net/pma/GSoC_2015_Mentoring_Organization_Application	Feb 2007 2008-2015
Perform Security Training		

- Document Technical Stack. The phpMyAdmin team documents their technical stack via a wiki page.
- Apply Secure Coding Standards. Over time, the phpMyAdmin project has adopted coding standards, and security coding standards, evidenced by the following series of statements taken from the developer email list:
 - August 2010: *Please stick with PEAR coding style and please try to keep your code as simple as possible: beginners are using phpMyAdmin as an example application.*
 - March 2012: *Here are some guidelines on how to avoid security issues that could lead to security bugs*
- Apply Security Tooling. Over time, the project has added tooling to support its standards and build process.¹⁸
 - May 2012: *To verify coding style you can install PHP_CodeSniffer and validate your code using PMAStandard file.php*
 - Sep 2013: *I've set up coveralls.io coverage reports*
 - Dec 2013: (r.e. analysis tool scrutinizer) *What is this good for?*
 - May 2014 : *Let's make scrutinizer happy ;).*
- Perform Security Testing. One of the projects leaders confirmed that external security consultants (researchers) are the primary finders of vulnerabilities. However, the team began developing an automated test suite in 2009, and continues to maintain and add to it.
 - May 2001: *It would maybe nice to have a kind of "test suite"*
 - Jul 2009: *I've set up a cronjob which would run the test suite e.g. every day*
 - Nov 2014 *We use several test suites to help ensure quality code. All new code should be covered by test cases to ensure proper [Unit Testing]*
- Perform Penetration Testing. External security researchers conduct penetration testing.
- Perform Security Review. The two main administrators monitor the build process and its associated tool reports for security as well as other quality attributes.

¹⁸<https://www.phpmyadmin.net/15-years/>

- **Publish Operations Guide.** The team maintains installation, configuration, and administration documentation for phpMyAdmin, including security-related material¹⁸.
- **Track Vulnerabilities.** The phpMyAdmin project maintains a security mailing address, and a list of vulnerabilities resolved. The team tracks and resolves vulnerabilities as they are reported.
- **Improve Development Process.** The phpMyAdmin team began participating in the Google Summer of Code in 2008. In conjunction with managing the additional help they received from the sponsored developers, phpMyAdmin administrators added or extended standards, tooling, and test suites to the development process.
- **Perform Security Training.** While we did not observe direct evidence of training, e.g. tutorials or courses, the project has brought developers in successfully through its mentoring program.

We present a display of the first occurrence of each practice, drawn from the sample of classified issues in Figure 8.3b.

8.4.2.3 Outcome Measures

We present phpMyAdmin CVE count per month (CVE), and as a running total (CVErt), as well as Vdensity and VRE for 2010-2016 in Figure 8.3c.

As described in Section 4.4.3, we have developed a set of measurements that we expect to capture security-related constructs for software development. In Table 8.3, we reprise the SP-EF data elements from Table 4.1, and give a summary of our data collection results for the four case study projects.

8.5 OpenSSL Case Study

OpenSSL is a widely-used library implementing Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols and cryptographic primitives¹⁹. OpenSSL provides an open defect repository based on and version control system, both based on a public Github repository²⁰, and the project issues security advisories and releases for critical security issues. In April 2014, a Google researcher discovered a critical vulnerability, now known as Heartbleed, in

¹⁹<https://www.openssl.org/>

²⁰<https://github.com/openssl/openssl>

(a) phpMyAdmin Context Factors

(b) phpMyAdmin Practice Adherence Keyword Counts

(c) phpMyAdmin Practice Adherence Outcome Measures

Figure 8.3 phpMyAdmin Metrics 2001-2014

Table 8.3 SP-EF Model Context Factors, Measures, and Case Study Results

Metric	Bitcoin	IBM wasCloud	phpMyAdmin	OpenSSL
Language	C, C++	Ruby, Java, Javascript	PHP, SQL, Javascript, and HTML	C
Operating System	Unix, Windows	Unix	Unix	Multiple
Domain	Online financial transactions	Web application utility	Web-based database administration	Secure online communications
Product Age	8 years	2 years	18 years	19 years
Source Lines of Code (SLOC)	200,000	100,000	500,000	500,000
Churn				
Team Size	10's	10's	10's	10's
Number of Machines	10,000's	1000's	1000's	1,000,000's
Number of Identities	100,000's	NA	NA	1,000,000's
Number of Dollars	1,000,000,000's	NA	NA	NA
Source Code Availability	Open	Closed	Open	Open
CIA Requirements	High/High/High	Low/High/High	High/High/Low	High/High/High
Team Location	Distributed	Distributed	Distributed	Distributed
Methodology	Agile	Agile	Agile	Agile
Perform Security Training	No	Yes	No	No
Apply Data Classification Scheme	No	Yes	No	No
Apply Security Requirements	Yes	Yes	Yes	Yes
Apply Threat Modeling	Yes	Yes	No	Yes
Document Technical Stack	Yes	Yes	Yes	Yes
Apply Secure Coding Standards	Yes	Yes	Yes	Yes
Apply Security Tooling	Yes	Yes	Yes	Yes
Perform Security Testing	Yes	Yes	Yes	Yes
Perform Penetration Testing	No	Yes	No	No
Perform Security Review	Yes	Yes	Yes	Yes
Publish Operations Guide	Yes	Yes	Yes	Yes
Track Vulnerabilities	Yes	Yes	Yes	Yes
Improve Development Process	Yes	Yes	Yes	Yes
Vulnerabilities	23	1	233	181
Defects	3,380	249	11,185	1353

OpenSSL code ²¹. In response to Heartbleed, the project team received an influx of developers and funding to apply to the project’s security ²². The OpenSSL case offers an opportunity for us to investigate whether SP-EF measurements reflect the changes made in security practice use where we know in advance that such changes exist.

8.5.1 Data Collection

We cloned the OpenSSL github repo, representing the source code and changes made during the history of the project. We applied CVSAly from the MetricsGrimoire tool set reported on by Gonzalez-Barahona [24] to process and summarize the github data, and then analyzed and reported on the data using R ²³. We extracted email data from downloaded developer email archives. We extracted defect and vulnerability data from OpenSSLs github repository ²⁴, and from the security section of its website <https://www.openssl.org/news/secadv/>, and checked it against the projects CVE records ²⁵.

For each artifact we identified, we recorded the document name, URL, age, and made note of any available change history, e.g. wiki page change records. We manually classified pre- and post-release vulnerabilities based on a study of who reported the vulnerability and whether they were identifiable as an OpenSSL developer (by whether the email address was associated with commits and with the discovery of the vulnerability described in the project’s reporting ²⁶).

8.5.2 Results

We present a summary of the SP-EF measurements for OpenSSL in Table 8.3. In this section, we present details of the SP-EF measurements for OpenSSL.

8.5.2.1 Context Factors

OpenSSL’s Domain is secure online communications, and the project is written in C. The context factors SLOC, Churn, Developers, and User Count evolved over time as shown in 8.3a. SLOC, Churn, and Developers are based on version control system data, while User Count is

²¹<http://heartbleed.com/>

²²<https://rwc.iacr.org/2017/Slides/rich.saltz.pdf>

²³<https://www.r-project.org/>

²⁴<https://github.com/openssl/openssl/issues>

²⁵http://www.cvedetails.com/vulnerability-list/vendor_id-217/product_id-383/Openssl-Openssl.html

²⁶<https://www.openssl.org/news/secadv/>

based on OpneHub data for the project ²⁷. We, subjectively, rate OpenSSL’s Confidentiality, Integrity, and Availability Requirements as High, because the software supports secure online communication.

8.5.2.2 Practice Adherence

In this section, we review security practice usage results obtained from researcher assessment of OpenSSL. We present observations organized by security practice, and include links to evidence, where available, in Table 8.4. Italicized quotes are from OpenSSL project communications.

Table 8.4 OpenSSL Manual Review Evidence

Security Practice	Source (Link)	Event Date
Apply Data Classification Scheme		
Apply Security Requirements	List of Standards https://www.openssl.org/docs/standards.html	
Apply Threat Modeling		
Document Technical Stack	Install: https://github.com/openssl/openssl/blob/master/INSTALL	
Apply Secure Coding Standards	https://www.openssl.org/policies/codingstyle.html	
Apply Security Tooling	https://github.com/openssl/openssl/blob/ff54cd9beb07e47c48dac02d3006b0fbe5fc6cc2/fuzz/README.md	
Perform Security Testing		
Perform Penetration Testing		
Perform Security Review		
Publish Operations Guide	Manual pages: https://www.openssl.org/docs/manpages.html Documentation: https://www.openssl.org/docs/ OpenSSL Cookbook: https://www.feistyduck.com/books/openssl-cookbook/	
Track Vulnerabilities		
Improve Development Process		
Perform Security Training		

- Apply Data Classification Scheme. We were not able to identify documented process or artifacts for data classification.
- Apply Security Requirements. Many of the protocols implemented by OpenSSL are described in Internet Engineering Task Force (IETF) Request for Comment (RFC) docu-

²⁷<https://www.openhub.net/p/openssl>

ments, for example RFC 5246 ²⁸ and RFC 6101 ²⁹. The team refers to RFC's in project communications, and in comments in the code. A version of OpenSSL has been certified as FIPS-140 compliant ³⁰.

- Apply Threat Modeling. We were not able to identify documented process or artifacts around threat modeling, however we identified threads of discussion on threats and potential threats in the project issues and emails. For example, *Basically, I'm asking for more considerations to be added to the threat mode, Only accept change cipher spec when it is expected instead of at any time. This prevents premature setting of session keys before the master secret is determined which an attacker could use as a MITM attack*
- Document Technical Stack. The developers describe the components required to build OpenSSL in the INSTALL document, and the stack can be further characterized through the contents of the project make and build files. The developers discuss updates to the stack via email and issue tracking.
- Apply Secure Coding Standards. The developers maintain a document describing how to contribute to the project, and a coding standards document.
- Apply Security Tooling. The developers fuzz test OpenSSL. We were not able to identify documented process for security tooling, however we found multiple mentions of Coveralls ³¹, a code coverage analyzer, in the issues, and in the test suite.
- Perform Security Testing. The OpenSSL project has begun requiring tests for each newly submitted change request. An automated test suite was added by project developers in April 2015.
- Perform Penetration Testing. We were not able to identify documented process or artifacts for penetration testing.
- Perform Security Review. As with Bitcoin, the developers manage changes to the codebase through Github Pull Requests. The contribution guidelines call for each pull request to be peer reviewed.

²⁸<https://tools.ietf.org/html/rfc5246>

²⁹<https://tools.ietf.org/html/rfc6101>

³⁰<https://www.openssl.org/docs/fips.html>

³¹<https://coveralls.io/>

- Publish Operations Guide. The development team maintains documentation for developers and users of OpenSSL, including security-related material.
- Track Vulnerabilities. The developers maintain a vulnerability reporting process, as well as a list of vulnerabilities resolved. The team tracks and resolves vulnerabilities as they are reported.
- Improve Development Process. Reports from both inside ³² and outside ³³ the team document process changes in response to Heartbleed.
- Perform Security Training. We were not able to identify documented process or artifacts for Security Training.

8.5.2.3 Outcome Measures

We present OpenSSL CVE count per month (CVE), and as a running total (CVErt), as well as Vdensity and VRE for 2010-2016 in Figure 8.4c.

8.6 Research Question Answers

In this section, we address the Research Questions using the data from the case studies.

8.6.1 RQ2.1: Can the complete set of SP-EF measurements be collected for a software development project?

Considering the four case studies summarized in Table 8.3, we make the following observations about data collection for the SP-EF measurements:

- Every measurement was recorded for at least one project.
- Usage measurements, for example, Number of Identities and Number of Machines, were unavailable for multiple projects.
- Apply Data Classification Scheme, Perform Penetration Testing, and Perform Security Training were conducted only on the closed source, commercial project at IBM.

We judge that the SP-EF measurements can, in principle, be collected for software development projects, however the necessary data is not always available.

³²<https://rwc.iacr.org/2017/Slides/rich.saltz.pdf>

³³<https://www.linux.com/blog/event/linuxcon-europe/2016/openssl-after-heartbleed>

(a) OpenSSL Context Factors

(b) OpenSSL Practice Adherence Keyword Counts

(c) OpenSSL Practice Adherence Outcome Measures

Figure 8.4 OpenSSL Metrics 2013-2016

8.6.2 RQ3: How does security practice adherence affect software development security outcomes?

To address RQ3, we present a qualitative chronological examination of changes in security practice adherence compared with changes in VDensity and VRE in the four case studies. We follow with a summary of trends in VDensity and VRE in relation to practice adoption and use.

8.6.2.1 Bitcoin

In our data (see Figure 8.4), five practices were introduced in late 2011; Perform Threat Modeling, Document Technical Stack, Apply Secure Coding Standards, and Improve Development Process. Bitcoin's first CVE's were reported in August and September of 2012, yielding a spike in VDensity. At the same time, VRE spiked to .20, indicating that one of five of the vulnerabilities were discovered internally by the team. In the three months that followed the first reported CVE's, we observed the first occurrences of Track Vulnerabilities, Perform Security Review, and Perform Security Testing. Two more groups of CVE's were reported in April and September of 2013. VRE rose in conjunction with the September vulnerabilities. VDensity plateaued over this period, and through most of 2014, as the amount of code added balanced out the increase in vulnerability count. VDensity spiked in October 2014 due to a decrease in SLOC, as no new vulnerabilities were reported in that month. Our first observations of Apply Security Requirements and Publish Operations Guide were in March and October 2015. We did not observe the use of Apply Data Classification Scheme, Perform Penetration Testing, or Perform Security Training.

IBM wasCloud In our data (see Figure 8.2), all of the security practices were introduced in mid-2014, in conjunction with an increase in the number of developers, the amount of code churn, and a spike in SLOC. The first vulnerability was reported in early 2015, and was caught internally by the team, yielding a spike in VDensity and a VRE of 1.00.

8.6.2.2 phpMyAdmin

In our data (see Figure 8.3), the first CVE was reported in 2001. We first observed Track Vulnerabilities in June 2003, VDensity increased through 2006. Perform Security Testing was first observed in November 2005. VRE spiked to .80 in December 2005, and then declined over the course of the period measured to 0.29. Improve Development Process was first observed in February 2007. Apply Secure Coding Standards and Document Technical Stack were first

observed in mid-2010. Apply Security Tooling was first observed in August 2011. VDensity plateaued near .20 through most of 2011 when it increased to .33 or more from October 2011 until May 2012, returning to the mid-20's in June 2012, and then declining over the period measured, reaching .09 in December 2015, the last month measured. We did not observe the use of Apply Data Classification Scheme, Apply Security Requirements, Perform Threat Modeling, Perform Penetration Testing, or Perform Security Training.

8.6.2.3 OpenSSL

In our data (see Figure 8.4), we observed use of Apply Security Requirements, Perform Threat Modeling, Document Technical Stack, Perform Security Review, Publish Operations Guide, Track Vulnerabilities, and Improve Development Process in the months preceding Heartbleed's discovery in April 2014. In April 2014, the practices Apply Secure Coding Standards, Apply Security Tooling, and Perform Security Testing were first observed. VDensity increased through mid-2015, with an increase in its slope after April 2014. VRE plateaued during the observation period until early 2015, when it rose to a higher plateau over the course of three months. Through examination of the commit records, we note that a test suite was added to the project in April 2015. We did not observe the use of Apply Data Classification Scheme, Perform Penetration Testing, or Perform Security Training.

8.6.2.4 Summary

We found examples of practice adoption preceding changes in VDensity and VRE in each project. We now discuss the VDensity and VRE trends we observed for each project, with reference to practice usage.

IBM wasCloud was in production for six months before a CVE was reported, and the team identified the vulnerability. The team used all of the SP-EF practices.

Bitcoin had the following practices in place at the time of its first CVEs and first increase in VRE: Perform Threat Modeling, Document Technical Stack, Apply Secure Coding Standards, and Improve Development Process. Bitcoin's VRE further increased after the introduction of Track Vulnerabilities, Perform Security Review, and Perform Security Testing.

phpMyAdmin experienced a multi-year decrease in VDensity with the following practices in place: Document Technical Stack, Apply Secure Coding Standards, Apply Security Tooling, Perform Security Testing, Perform Security Review, Publish Operations Guide, Track Vulnerabilities, and Improve Development Process. phpMyAdmin experienced an increase in VRE in the months after the introduction of Perform Security Testing. OpenSSL experienced an

Figure 8.5 Vulnerability Density (VDensity) and Vulnerability Removal Effectiveness (VRE) Boxplots for Case Study Projects

increase in VRE on the introduction of a test suite.

OpenSSL experienced a decrease in VDensity and an increase in VRE with the following practices in place: Apply Security Requirements, Perform Threat Modeling, Document Technical Stack, Apply Secure Coding Standards, Apply Security Tooling, Perform Security Testing, Perform Security Review, Publish Operations Guide, Track Vulnerabilities, and Improve Development Process.

Figure 8.5 presents the Adherence, VDensity and VRE ranges observed in the four case study projects. The project with the highest performance (lowest VDensity, highest VRE), IBM wasCloud, used all practices before the first vulnerability. However, IBM wasCloud was also the smallest, and newest of the four projects. Reviewing the project's progress over time may reveal variations in the outcomes. Bitcoin had the highest median Adherence, second lowest VDensity, and ranked third in median VRE. We note that the pattern of the Bitcoin development process reflects steady or declining VDensity, and increasing VRE. phpMyAdmin ranked second in median Adherence, second to last in VDensity, and second highest in median VRE. VDensity reflects the overall history of the project, and phpMyAdmin has seen decreasing VDensity in recent years, reflecting the team's adoption of practices over time. OpenSSL had the second lowest median Adherence, highest median VDensity, and the lowest VRE.

8.7 Discussion

In this section, we make additional observations, and discuss the implications of our results.

We observe three cross-project trends, and offer them as recommendations:

- **Perform Security Testing to find vulnerabilities** We observed that VRE increased after the adoption of Perform Security Testing in both phpMyAdmin and OpenSSL. In previous work [68], we found that Functional Testing was associated with increases in vulnerability discovery in Chrome, Firefox, and phpMyAdmin.
- **Apply Security Tools to vulnerabilities** We observed that VDensity increased within months after the adoption of Apply Security Tooling in both phpMyAdmin and OpenSSL.
- **Adopt core practices to lower vulnerability density** We observed that the practices common to projects that experienced reduction in VDensity include: Document Technical Stack, Apply Secure Coding Standards, Apply Security Tooling, Perform Security Testing, Perform Security Review, Publish Operations Guide, Track Vulnerabilities, and Improve Development Process. While all of the listed practices were present, reduction in VDensity did not always occur immediately, and, in some cases, increased before declining.
- **Vulnerabilities Take Time** Anywhere from six months to three years preceded the first reported CVEs for the projects we studied. The absence of a reported vulnerability may reflect low usage or a lack of effort in vulnerability discovery rather than high security, so VDensity must be examined in context.
- **Solutions Take Time** While the introduction of practices have been associated with lower VDensity and higher VRE in a matter of months in some cases, declines in VDensity were also seen after a set of practices were in place for six months or more.
- **Crisis Prompts Action** In our data, a reported CVE preceded security practice adoption.
- **Sometimes things appear worse before they get better** In our data, the introduction of vulnerability discovery practices to large codebases was associated with increased VDensity, followed by a later decline.

8.8 Limitations

Because the development team members are the ground truth source for security practice use, researcher assessment and classification cannot make a fully accurate report of practice use. To address subjectivity concerns, we have been systematic in selecting the security practices to include in the framework, and in identifying the framework practices in the projects we studied. We have considered a limited pool of security practices, drawn from four existing lists of security practices. Whether these encompass security needs in other contexts is unknown. We evaluated both large and small projects as a check against this limitation. Our source for practice use, project issue tracking records, is an incomplete record of security practice use. For example, from a previous case study, we [68] know that phpMyAdmin collaborates with external security researchers to conduct penetration testing of the software before its release, a fact that did not emerge in the data collected and analyzed here. On the other hand, our findings for OpenSSL are confirmed descriptions from the project team of the changes they made after Heartbleed ³⁴. Examination and evaluation of other project communications sources are necessary to fully validate our approach.

8.9 Conclusion

We have applied SP-EF to collect longitudinal data for Bitcoin, IBM wasCloud, phpMyAdmin, and OpenSSL. In our results, we found evidence that increased security practice use is associated with lower vulnerability density, and with higher vulnerability removal effectiveness. We observed that increased team size was associated with increased security practice use, although further study is necessary to identify the reasons behind the association. Replications of SP-EF case studies are needed to identify whether these findings are generalizable, and to check the reliability of the framework.

³⁴<https://rwc.iacr.org/2017/Slides/rich.saltz.pdf>

Chapter 9

Conclusion

As society increases its dependence on software for the storage, transmission and use of sensitive data, the need for software to do so in a secure way grows correspondingly. Understanding how to prevent vulnerabilities for sensitive data requires an understanding of all aspects of the software and systems that store, transmit, and operate on that data. Preventing vulnerabilities in software after its release requires attention to security while the software is being developed. For development teams to understand the sources of vulnerabilities and arrange for their prevention, they must have access to records of what has worked, and what has failed, in the past. This research has been conducted under the premise that we can improve the state of these necessary records for the benefit of software development teams. Our thesis is that measuring the effect of security practice adherence on security outcomes requires accounting for software development context factors and software usage context factors.

To support measurement of security practice use and outcomes in software development, we have proposed and evaluated a measurement framework, SP-EF. In SP-EF, we have produced a set of measurements of software development projects that have a theoretical basis for being predictive of software security attributes. In SOTM, we have built an assessment framework for SP-EF, allowing estimation of how much impact each measurement has on software security outcomes, and allowing estimation of the importance of each security practice to a software's security outcomes.

We conducted literature reviews to identify software development security context factors, security practices, and outcome measures, reported in Section 3. We identified a set of security practices, practice adherence metrics, outcome measures, and context factors, and assembled them as a measurement framework, SP-EF, reported on in Section 4. We conducted a case study in which we collected all SP-EF data from an industrial software development project,

reported on in Section 5. We found agreement between the researcher and team views of security practice use on the project, and evaluated the effectiveness of automated means of assessing practice adherence. We identified use of all of the practices specified in SP-EF by one or more survey participants. We conducted a survey of open source development projects to assess the use of the SP-EF security practices and the degree to which our adherence measures correlate with security practice use, reported on in Section 6. We found empirical support for the use of the SP-EF security practices. We found that Training has a positive, statistically significant correlation with Usage, suggesting that investment in training supports practice usage. We assessed whether the theorized relationships in SOTM hold in observational data by combining SP-EF measurements available in published datasets, reported on in Section 7. Our data suggest that not only software attributes, but the context of software use must be accounted for to assess the state of software security. Finally, we conducted longitudinal studies of four software projects, finding that, in our data, testing and tooling support increased vulnerability discovery, and that a collection of practices is associated with a pattern of decrease in vulnerability density.

9.1 Contributions

- Security Outcomes Theoretical Model (SOTM) We defined a set of constructs relating software development and context factors to security outcomes, and observed the theorized relationships in historical records of project data.
- SP-EF We proposed a set of software development context factors, practice adherence measures, and context factors.
- A list of software development security practices that are descriptive of the security efforts of software development teams, and found empirical evidence of their use through a survey of open source development teams.
- a set of practice adherence measures based on technology acceptance theory.
- A security practice adherence survey questionnaire
- In the OpenHub-NVD data, Usage Risk, as measured by user count, has a comparable correlation with Outcomes to Software Risk, as measured by SLOC, Churn, Contributor Count, and project age. Factors outside the team’s direct influence have to be considered when evaluating security performance and mitigations.

- Our data corroborate previous research findings that team size, SLOC, and code churn are correlated with manifest vulnerabilities, and do so while controlling for other factors influencing security Outcomes. Measuring the relative impact of each measurement on its construct, and on the model's performance as a whole, supports refining the measurement framework and the theoretical model as further data are collected and evaluated.

9.2 Future Work

In conducting future work, we seek to exploit three benefits of the model we have presented and evaluated:

- SEM fit indicators and parameter estimates provide numerical assessment of model and metric performance, enabling data-driven assessment and iterative improvement of metric use. Collecting the current set of measurements for further projects will help to characterize suitable security practices for the project contexts studied
- The structural model constructs provide guidance for the kind of measurements to collect when evaluating software security. Collecting additional variables and assessing their impacts on model fit and outcome prediction supports iterative improvement of the model, and of software security in the projects that apply lessons learned from the model.
- Because the structural model does not depend on a particular set of measurements, it can be applied at other granularities than the software development project example used in this paper. We seek to apply SOTM to evaluating binaries, source files, and commits, in observational studies, in controlled experiments, and in action research.

BIBLIOGRAPHY

- [1] Alhazmi, O. H. et al. “Measuring, analyzing and predicting security vulnerabilities in software systems”. *Computers & Security* **26.3** (2007), pp. 219–228.
- [2] Anderson, R. “Security in Open versus Closed Systems - The Dance of Boltzmann, Coase and Moore”. *Proceedings of 2002 Open Source Software : Economics, Law and Policy*. Toulouse, France: IEEE Press, 2002, pp. 4–14.
- [3] Austin, A. & Williams, L. “One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques”. *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*. ESEM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 97–106.
- [4] Austin, A. et al. “A comparison of the efficiency and effectiveness of vulnerability discovery techniques”. *Information and Software Technology* **55.7** (2013), pp. 1279–1288.
- [5] Ayalew, T. et al. *Identification and evaluation of security activities in agile projects*. Springer, 2013, 139153.
- [6] Baca, D. & Carlsson, B. “Agile Development with Security Engineering Activities”. *Proceedings of the 2011 International Conference on Software and Systems Process*. ICSSP 2011. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 149–158.
- [7] Basili, V. et al. “Building knowledge through families of experiments”. *Software Engineering, IEEE Transactions on* **25.4** (1999), pp. 456–473.
- [8] Borsboom, D. “Latent Variable Theory”. *Measurement: Interdisciplinary Research and Perspectives* **6.1-2** (2008), pp. 25–53. eprint: <http://dx.doi.org/10.1080/15366360802035497>.
- [9] Borsboom, D. M. “The theoretical status of latent variables”. *Psychological Review* **110.2** (2003), pp. 203–209.
- [10] Budgen, D. et al. “Using mapping studies in software engineering”. *Proceedings of PPIG*. Vol. 8. 2008, pp. 195–204.
- [11] Camilo, F. et al. “Do Bugs Foreshadow Vulnerabilities?: A Study of the Chromium Project”. *Proceedings of the 12th Working Conference on Mining Software Repositories*. MSR '15. Florence, Italy: IEEE Press, 2015, pp. 269–279.
- [12] Capra, E. et al. “An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects”. *IEEE Trans. Softw. Eng.* **34.6** (2008-11), pp. 765–782.

- [13] Catal, C. & Diri, B. “Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem”. *Information Sciences* **179.8** (2009), pp. 1040 –1058.
- [14] Dashevskiy, S. et al. “On the Security Cost of Using a Free and Open Source Component in a Proprietary Product”. *Engineering Secure Software and Systems: 8th International Symposium, ESSoS 2016, London, UK, April 6–8, 2016. Proceedings*. Cham: Springer International Publishing, 2016, pp. 190–206.
- [15] Davis, F. *A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results*. Cambridge, MA, 1986.
- [16] Davis, F. D. “Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology”. *MIS Q.* **13.3** (1989), pp. 319–340.
- [17] Dowd, M. et al. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006.
- [18] Epstein, J. “A Survey of Vendor Software Assurance Practices”. *Proceedings of the 2009 Annual Computer Security Applications Conference. ACSAC '09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 528–537.
- [19] Fenton, N. & Neil, M. *Risk Assessment and Decision Analysis with Bayesian Networks*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 2012.
- [20] Fenton, N. E. & Pfleeger, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2nd. Boston, MA, USA: PWS Publishing Co., 1998.
- [21] *Four Grand Challenges in Trustworthy Computing*. http://cra.org/uploads/documents/resources/rissues/trustworthy.computing_.pdf. 2003.
- [22] Fitcher, L. & Solms, R. von. “Guidelines for Secure Software Development”. *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*. SAICSIT '08. ACM. Wilderness, South Africa, 2008, pp. 56–65.
- [23] Glaser, B. G. & Strauss, A. L. *The Discovery of Grounded Theory; Strategies for Grounded Research*. New York, NY: Aldine de Gruyter, 1967.
- [24] Gonzalez-Barahona, J. M. et al. “The MetricsGrimoire Database Collection”. *Proceedings of the 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. MSR '15*. Washington, DC, USA: IEEE Computer Society, 2015, pp. 478–481.

- [25] Gopal, A. et al. “The impact of institutional forces on software metrics programs”. *IEEE Transactions on Software Engineering* **31.8** (2005-08), pp. 679–694.
- [26] Harris, E. & Perlroth, N. “For Target, the Breach Numbers Grow”. *New York Times* (10, 2014).
- [27] Hildreth, L. *Residual analysis for structural equation modeling*. 2013.
- [28] Howard, M. & Lipner, S. *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006.
- [29] Howard, M. & Lipner, S. *The security development lifecycle*. O’Reilly Media, Incorporated, 2009.
- [30] Hudson, W. “Card Sorting” (2013).
- [31] “IEEE Standard Glossary of Software Engineering Terminology”. *IEEE Std 610.12-1990* (1990), pp. 1–84.
- [32] Jensen, W. *Directions in Security Metrics Research*. <http://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7564.pdf>. 2009.
- [33] Jones, C. *Software Assessments, Benchmarks and Best Practices*. Boston, MA: Addison Wesley, 2000.
- [34] Kaminsky, D. et al. *Showing How Security Has (And Hasn’t) Improved, After Ten Years of Trying*. <http://www.slideshare.net/dakami/showing-how-security-has-and-hasnt-improved-after-ten-years-of-trying>. 2011.
- [35] Kan, S. H. *Metrics and Models in Software Quality Engineering*. 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [36] Kitchenham, B. A. et al. “Towards an ontology of software maintenance”. *Journal of Software Maintenance: Research and Practice* **11.6** (1999), pp. 365–389.
- [37] Kline, R. B. *Principles and Practice of Structural Equation Modeling*. 4th. Guilford Publications, 2015. 534 pp.
- [38] Kocaguneli, E. et al. “Distributed Development Considered Harmful?” *Proceedings of the 2013 International Conference on Software Engineering*. ICSE ’13. San Francisco, CA, USA: IEEE Press, 2013, pp. 882–890.
- [39] Krebs, W. “Turning the Knobs: A Coaching Pattern for XP through Agile Metrics”. *Proceedings of the Extreme Programming and Agile Methods XP/Agile Universe 2002*. XP/Agile ’02. ACM. Chicago, IL, 2002, pp. 60–69.

- [40] Krsul, I. V. “Software vulnerability analysis”. PhD thesis. West Lafayette, IN, USA: Purdue University, 1998.
- [41] Lamkanfi, A. et al. “The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information”. *Mining Software Repositories (MSR)*, 2013 10th IEEE Working Conference on. 2013-05, pp. 203–206.
- [42] Layman, L. “Essential communication practices for Extreme Programming in a global software development team”. *Information and Software Technology* **48.9** (2006). Special Issue Section: Distributed Software Development, pp. 781 –794.
- [43] Layman, L. et al. “Motivations and Measurements in an Agile Case Study”. *J. Syst. Archit.* **52.11** (2006), pp. 654–667.
- [44] Legris, P. et al. “Why Do People Use Information Technology?: A Critical Review of the Technology Acceptance Model”. *Inf. Manage.* **40.3** (2003), pp. 191–204.
- [45] Lennon, E. E. *IT Security Metrics*. *ITL Bulletin*. <http://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7564.pdf>. 2003.
- [46] Loehlin John, C. *Latent Variable Models: An Introduction to Factor, Path, and Structural Analysis*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1986.
- [47] Loehlin, J. C. *Latent Variable Models: An Introduction to Factor, Path, and Structural Equation Analysis*. 4th. Guilford Publications, 2004. 317 pp.
- [48] M, K. “Smoking: The cancer controversy. some attempts to assess the evidence.” *Archives of Internal Medicine* **112.3** (1963), pp. 448–450. eprint: [/data/journals/intemed/15500/archinte_112_3_042.pdf](http://data/journals/intemed/15500/archinte_112_3_042.pdf).
- [49] Martinez, M. & Pellegrino, M. *OWASP Software Security Assurance Process*. https://www.owasp.org/index.php/OWASP_Software_Security_Assurance_Process#tab=Main.
- [50] McConnell, S. *Code Complete*, 2nd ed. Redmond, WA: Microsoft Press, 2004.
- [51] McGraw, G. et al. *The Building Security In Maturity Model*. www.bsimm.com. 2013.
- [52] McGraw, G. *Software security: Building Security In*. Addison-Wesley Professional, 2006.
- [53] McIntosh, S. et al. “The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects”. *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 192–201.

- [54] Mell, P. et al. “Common Vulnerability Scoring System”. *Security Privacy, IEEE* **4.6** (2006-11), pp. 85–89.
- [55] Mell, P. et al. “A complete guide to the common vulnerability scoring system version 2.0”. *Published by FIRST-Forum of Incident Response and Security Teams*. 2007, pp. 1–23.
- [56] Members, T. C.C.R. A. *Common Criteria for Information Technology Security Evaluation, Par 1: Introduction and geeral model*. 2012.
- [57] Meneely, A. et al. “When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits”. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 2013, pp. 65–74.
- [58] Meneely, A. et al. “Validating Software Metrics: A Spectrum of Philosophies”. *ACM Trans. Softw. Eng. Methodol.* **21.4** (2013-02), 24:1–24:28.
- [59] Meneely, A. et al. “An Empirical Investigation of Socio-technical Code Review Metrics and Security Vulnerabilities”. *Proceedings of the 6th International Workshop on Social Software Engineering*. SSE 2014. Hong Kong, China: ACM, 2014, pp. 37–44.
- [60] Menzies, T. et al. *Security cannot be measured*. Morgan Kaufman, 2016.
- [61] Morrison, P. *A Security Practices Evaluation Framework*. <https://pjmorris.github.io/Security-Practices-Evaluation-Framework/guidebook.html>.
- [62] Morrison, P. *A Security Practices Evaluation Framework*. <https://pjmorris.github.io/Security-Practices-Evaluation-Framework/guidebook.html>.
- [63] Morrison, P. *A Security Practices Evaluation Framework*. <https://pjmorris.github.io/Security-Practices-Evaluation-Framework/>, 2016.
- [64] Morrison, P. et al. “Challenges with Applying Vulnerability Prediction Models”. *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*. HotSoS ’15. Urbana, Illinois: ACM, 2015, 4:1–4:9.
- [65] Morrison, P. et al. “Mapping the Field of Software Development Security Metrics”. *Submitted to Information and Software Technology* (2017).
- [66] Morrison, P. et al. “Measuring Security Practice Use: A Case Study At IBM”. *Proceedings of the 5th Annual Workshop on Conducting Empirical Studies in Industry*. CESI ’17. Buenos Aires, Argentina: ACM, 2017.

- [67] Morrison, P. et al. “Surveying Security Practice Adherence in Software Development”. *Proceedings of the 2017 Symposium and Bootcamp on the Science of Security*. HotSoS ’17. Hanover, MD: ACM, 2017.
- [68] Morrison, P. J. et al. “Are vulnerabilities discovered and resolved like other defects?” *Empirical Software Engineering* (2017).
- [69] Nagappan, M. et al. “Diversity in Software Engineering Research”. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2013. Saint Petersburg, Russia: ACM, 2013, pp. 466–476.
- [70] Nakamoto, S. *Bitcoin: A peer-to-peer electronic cash system*, <http://bitcoin.org/bitcoin.pdf>. 2008.
- [71] Ouedraogo, M. et al. “Appraisal and Reporting of Security Assurance at Operational Systems Level”. *J. Syst. Softw.* **85**.1 (2012), 193208.
- [72] Oyetoyan, T. D. et al. “An Empirical Study on the Relationship between Software Security Skills, Usage and Training Needs in Agile Settings”. *2016 11th International Conference on Availability, Reliability and Security (ARES)*. 2016, pp. 548–555.
- [73] Ozment, J. A. “Vulnerability discovery & software security”. PhD thesis. Citeseer, 2007.
- [74] P., P. C. *Security in computing*. Prentice-Hall, 1997.
- [75] Pfleeger, S. & Cunningham, R. “Why Measuring Security Is Hard”. *Security Privacy, IEEE* **8**.4 (2010), pp. 46–54.
- [76] Pham, N. et al. “A Near Real-Time System for Security Assurance Assessment”. *Proceedings of the 2008 The Third International Conference on Internet Monitoring and Protection*. ICIMP ’08. Washington, DC, USA: IEEE Computer Society, 2008, 152160.
- [77] Potter, J. D. *The Lady Tasting Tea: How Statistics Revolutionized Science in the Twentieth Century*. Holt and Company, 2002.
- [78] Ray, B. et al. “A Large Scale Study of Programming Languages and Code Quality in Github”. *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. Hong Kong, China: ACM, 2014, pp. 155–165.
- [79] Riaz, M. et al. “Hidden in plain sight: Automatically identifying security requirements from natural language artifacts”. *Proc. 22nd RE*. IEEE. 2014, pp. 183–192.

- [80] Roseel, Y. “lavaan: An R Package for Structural Equation Modeling”. *Journal of Statistical Software* (2012), pp. 1–36.
- [81] Rudolph, M. & Schwarz, R. “A Critical Survey of Security Indicator Approaches”. *Proceedings of the 2012 Seventh International Conference on Availability, Reliability and Security*. Conference on Availability, Reliability and Security. Washington, DC, USA, 2012, pp. 291–300.
- [82] Savola, R. M. “Quality of security metrics and measurements”. *Computers & Security* **37** (2013), pp. 78 –90.
- [83] Schneidewind, N. F. “Methodology for Validating Software Metrics”. *IEEE Trans. Softw. Eng.* **18.5** (1992), 410422.
- [84] Schreier, M. *Qualitative Content Analysis in Practice*. New Delhi: SAGE Publications, 2012.
- [85] Schultz, J. E. et al. *Responding to computer security incidents*. <ftp://ftp.cert.dfn.de/pub/docs/csir/ihg.ps.gz>; . 1990.
- [86] Shin, Y. et al. “Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities”. *IEEE Transactions on Software Engineering* **37.6** (2011), pp. 772–787.
- [87] Shirey, R. *Internet Security Glossary, Version 2*. Request for Comments 4949. Published: RFC 4949 (Informational). IETF, 2007.
- [88] Shostack, A. *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.
- [89] Simpson, S. et al. *Fundamental Practices for Secure Software Development*. http://www.safecode.org/publication/SAFECode_Dev_Practices0211.pdf. 2013.
- [90] Thongtanunam, P. et al. “Review participation in modern code review”. *Empirical Software Engineering* (2016), pp. 1 –50.
- [91] Uzunov, A. V. et al. “A comprehensive pattern-oriented approach to engineering security methodologies”. *Information and Software Technology* **57** (2015), pp. 217 –247.
- [92] Venkatesh, V. et al. “User Acceptance of Information Technology: Toward a Unified View”. *MIS Q.* **27.3** (2003), pp. 425–478.
- [93] Verendel, V. “Quantified security is a weak hypothesis: a critical survey of results and assumptions”. *Proceedings of the 2009 workshop on New security paradigms*. ACM. 2009, pp. 37–50.

- [94] Walden, J. et al. "Idea: Java vs. PHP: Security Implications of Language Choice for Web Applications". *Proceedings of the Second International Conference on Engineering Secure Software and Systems*. ESSoS'10. Pisa, Italy: Springer-Verlag, 2010, pp. 61–69.
- [95] Wallace, L. G. & Sheetz, S. D. "The adoption of software measures: A technology acceptance model (TAM) perspective". *Information & Management* **51.2** (2014), pp. 249–259.
- [96] Wasserman, L. *All of Statistics*. Springer, 2004.
- [97] Williams, L. et al. *Evaluation Framework for Object-Oriented Languages: Version 1.4*. 2004.
- [98] Williams, L. et al. "Toward a Framework for Evaluating Extreme Programming". *Proceedings, Empirical Assessment in Software Engineering (EASE) 2004*. EASE '04. ACM. Edinburgh, UK, 2004.
- [99] Williams, M. D. et al. "The unified theory of acceptance and use of technology (UTAUT): A literature review". *Journal of Enterprise Information Management* **28.3** (2015), pp. 443–488.
- [100] Win, B. D. et al. "On the secure software development process: CLASP, {SDL} and Touchpoints compared". *Information and Software Technology* **51.7** (2009), pp. 1152–1171.
- [101] Wohlin, C. et al. *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
- [102] Wood, S. et al. "Successful Extreme Programming: Fidelity to the methodology or good teamworking?" *Inf. Softw. Technol.* **55.4** (2013), pp. 660–672.
- [103] Zhang, F. et al. "Towards Building a Universal Defect Prediction Model". *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 182–191.
- [104] Zhang, H. et al. "Identifying relevant studies in software engineering". *Information and Software Technology* **53.6** (2011). Special Section: Best papers from the {APSECBest} papers from the {APSEC}, pp. 625–637.
- [105] Zimmermann, T. et al. "Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista". *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*. 2010, pp. 421–428.

APPENDICES

Appendix A

Selected Papers

Table A.1 Selected Papers

Paper Id	Paper
P1	Alshammari, Bandar; Fidge, Colin; Corney, Diane, A Hierarchical Security Assessment Model for Object-Oriented Programs, 2011
P2	Gonzalez, R.M.; Martin, M.V.; Munoz-Arteaga, J.; Alvarez-Rodriguez, F.; Garcia-Ruiz, M.A., A measurement model for secure and usable e-commerce websites, 2009
P3	Pham, N.; Baud, L.; Bellot, P.; Riguidel, M., A near real-time system for security assurance assessment, 2008
P4	Hajdarevic, K.; Allen, P., A new method for the identification of proactive information security management system metrics, 2013
P5	Xueqi Cheng; Nannan He; Hsiao, M.S., A New Security Sensitivity Measurement for Software Variables, 2008
P6	Yanguo Liu; Traore, I.; Hoole, A.M., A Service-Oriented Framework for Quantitative Security Analysis of Software Architectures, 2008
P7	Marconato, G.V.; Kaniche, M.; Nicomette, V., A vulnerability life cycle-based security modeling and evaluation approach. 2012
P8	Scarfone, Karen; Mell, Peter, An analysis of CVSS version 2 vulnerability scoring, 2009
P9*	Sen-Tarng Lai, An Analyzer-Based Software Security Measurement Model for Enhancing Software System Security, 2010
P10	Wang, Lingyu; Islam, Tania; Long, Tao; Singhal, Anoop; Jajodia, Sushil, An Attack Graph-Based Probabilistic Security Metric, 2008
P11*	Manadhata, P.K.; Wing, J.M., An Attack Surface Metric, 2011
P12	Agrawal, Alka; Chandra, Shalini; Khan, Raees Ahmad, An efficient measurement of object oriented design vulnerability, 2009
P13	Wang, Ruyi; Gao, Ling; Sun, Qian; Sun, Deheng, An Improved CVSS-based Vulnerability Scoring Mechanism, 2011
P14	Wright, Jason L.; McQueen, Miles; Wellman, Lawrence, Analyses of Two End-User Software Vulnerability Exposure Metrics, 2012
P15*	Ouedraogo, Moussa; Khadraoui, Djamel; Mouratidis, Haralambos; Dubois, Eric, Appraisal and reporting of security assurance at operational systems level, 2012
P16	Sharma, Vibhu Saujanya; Trivedi, Kishor S., Architecture Based Analysis of Performance, Reliability and Security of Software Systems, 2005
P17*	Almorsy, M.; Grundy, J.; Ibrahim, A.S., Automated software architecture security risk analysis using formalized signatures, 2013
P18	Chowdhury, I.; Zulkernine, M., March. Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?, 2010
P19	Sultan, K.; En-Nouaary, A; Hamou-Lhadj, A, Catalog of Metrics for Assessing Security Risks of Software throughout the Software Development Life Cycle, 2008

Table A.2 Selected Papers

Paper Id	Paper
P20	Kanzaki, Y.; Igaki, H.; Nakamura, M.; Monden, A.; Matsumoto, K.I., January. Characterizing dynamics of information leakage in security-sensitive software process., 2005
P21	Mell, P.; Scarfone, K.; Romanosky, S., Common Vulnerability Scoring System, 2006
P22	Yanguo Liu; Traore, I., Complexity Measures for Secure Service-Oriented Software Architectures, 2007
P23	Walton, G.H.; Longstaff, T.A.; Linger, R.C., Computational Evaluation of Software Security Attributes, 2009
P24	Aissa, A.B.; Abercrombie, R.K.; Sheldon, F.T.; Mili, A., Defining and computing a value based cyber-security measure., 2012
P25*	Yonghee Shin; Meneely, A.; Williams, L.; Osborne, J.A., Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities, 2011
P26	Krautsevich L.; Martinelli F.; Yautsiukhin A., Formal approach to security metrics.: What does more secure mean for you?, 2010
P27	Michael, J.B.; Shing, Man-Tak; Cruickshank, K.J.; Redmond, P.J., Hazard Analysis and Validation Metrics Framework for System of Systems Software Safety, 2010
P28	Walden J.; Doyle M.; Lenhof R.; Murray J., Idea: java vs. PHP: security implications of language choice for web applications., 2010
P29	Vijayaraghavan, V.; Paul, S., iMeasure Security (iMS): A Novel Framework for Security Quantification, 2009
P30	Khan, S.A.; Khan, R.A., Integrity quantification model for object oriented design., 2012
P31	Liu, M.Y.; Traore, I., Measurement Framework for Software Privilege Protection Based on User Interaction Analysis, 2005
P32	Hasle, H.; Kristiansen, Yngve; Kintel, Ketil; Snekenes, Einar, Measuring Resistance to Social Engineering, 2005
P33	Islam, S.; Falcarin, P., Measuring security requirements for software security, 2011
P34	Manadhata, Pratyusa; Wing, Jeannette; Flynn, Mark; McQueen, Miles, Measuring the Attack Surfaces of Two FTP Daemons, 2006
P35	Buyens, Koen; Scandariato, Riccardo; Joosen, Wouter, Measuring the interplay of security principles in software architectures, 2009
P36	Alhazmi, O. H.; Malaiya, Y. K.; Ray, I., Measuring, analyzing and predicting security vulnerabilities in software systems, 2007
P37	Huaijun Wang; Dingyi Fang; Ni Wang; Zhanyong Tang; Feng Chen; Yuanxiang Gu, Method to Evaluate Software Protection Based on Attack Modeling, 2013
P38	Villarrubia, C.; Fernandez-Medina, E.; Piattini, M., Metrics of password management policy., 2006

Table A.3 Selected Papers

Paper Id	Paper
P39	Shar, Lwin Khin; Hee Beng Kuan Tan, Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities, 2012
P40	LeMay, E.; Ford, M.D.; Keefe, K.; Sanders, W.H.; Muehrcke, C., Model-based Security Metrics Using ADversary View Security Evaluation (ADVISE), 2011
P41	Gallon, Laurent, On the Impact of Environmental Metrics on CVSS Scores, 2010
P42	Schryen, G.; Kadura, R., Open source vs. closed source software: towards measuring security., 2009
P43	Bhme, R.; Flegyhzi, M., Optimal information security investment with penetration testing., 2010
P44	Gegick, M.; Rotella, P.; Williams, L., Predicting Attack-prone Components, 2009
P45	Shar, Lwin Khin; Tan, Hee Beng Kuan, Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns, 2013
P46	Nguyen, Viet Hung; Tran, Le Minh Sang, Predicting Vulnerable Software Components with Dependency Graphs, 2010
P47	Gegick, Michael; Williams, Laurie; Osborne, Jason; Vouk, Mladen, Prioritizing Software Security Fortification Through code-level Metrics, 2008
P48	Khan, M.U.A.; Zulkernine, M., Quantifying Security in Secure Software Development Phases, 2008
P49	Sharma, Vibhu Saujanya; Trivedi, Kishor S., Quantifying software performance, reliability and security: An architecture-based approach, 2007
P50	Arnold, F.; Pieters, W.; Stoelinga, M., Quantitative penetration testing with item response theory, 2013
P51	Grunske, Lars; Joyce, David, Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles, 2008
P52	Li, Minglu; Li, Jianping; Song, Hao; Wu, Dengsheng, Risk Management in the Trustworthy Software Process: A Novel Risk and Trustworthiness Measurement Model Framework, 2009
P53	Zimmermann, Thomas; Nagappan, Nachiappan; Williams, Laurie, Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista, 2010
P54*	Wang, Ju An; Wang, Hao; Guo, Minzhe; Xia, Min, Security Metrics for Software Systems, 2009
P55	Chowdhury, Istehad; Chan, Brian; Zulkernine, Mohammad, Security Metrics for Source Code Structures, 2008
P56	Trek, D., Security Metrics Foundations for Computer Security., 2009
P57	Abdulrazeg, A.A.; Norwawi, N.M.; Basir, N., Security metrics to improve misuse case model, 2012
P58	Walden, J.; Doyle, M.; Welch, G.A.; Whelan, M., Security of open source web applications, 2009

Table A.4 Selected Papers

Paper Id	Paper
P59	Demme, J.; Martin, R.; Waksman, A.; Sethumadhavan, S., Side-channel vulnerability factor: A metric for measuring information leakage, 2012
P60	Meneely, A.; Williams, L., Strengthening the empirical analysis of the relationship between Linus' Law and software security., 2010
P61*	Davidson, M.A., The Good, the Bad, And the Ugly: Stepping on the Security Scale, 2009
P62	Gegick, Michael; Rotella, Pete; Williams, Laurie, Toward Non-security Failures As a Predictor of Security Faults and Failures, 2009
P63	Neto, A.A.; Vieira, M., Trustworthiness Benchmarking of Web Applications Using Static Code Analysis, 2011
P64	Younis, A.A.; Malaiya, Y.K.; Ray, I., Using Attack Surface Entry Points and Reachability Analysis to Assess the Risk of Software Vulnerability Exploitability, 2014
P66	Beres, Y.; Mont, Marco Casassa; Griffin, J.; Shiu, S., Using security metrics coupled with predictive modeling and simulation to assess security processes, 2009
P67	Younis, A.A.; Malaiya, Y.K., Using Software Structure to Predict Vulnerability Exploitation Potential, 2014
P68	Perl, H.; Dechand, S.; Smith, M.; Arp, D.; Yamaguchi, F.; Rieck, K.; Fahl, S.; Acar, Y., VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits., 2015
P69*	Jinyoo Kim; Malaiya, Y.K.; Indrakshi Ray, Vulnerability Discovery in Multi-Version Software Systems, 2007
P70	Scarfone, K.; Mell, P., Vulnerability scoring for security configuration settings., 2008
P71	Meneely, A.; Srinivasan, H.; Musa, A.; Rodriguez Tejeda, A.; Mokary, M.; Spates, B., When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits, 2013

Appendix B

Survey Questionnaire

We built a survey questionnaire on the Qualtrics survey service¹, based on the list of security practices we developed, and the survey questions and scales described in 4.1. We included demographic questions, and a page of security practice definitions in the questionnaire. In addition to the closed questions based on the SP-EF practice adherence measures, we included open questions to probe developer views on the security practices.

To strike a balance between comprehensiveness and brevity of the survey, we selected the 13 most-referenced practices (Table 3.11 is ordered by number of references to each practice) for the survey. The practices not selected account for less than 2% of references to practices in our sources.

Four of the subjective adherence measures are actionable; comparing observed results with expected results can be translated to actions for each measure in the following ways;

- Usage - “How often is this practice applied?” - Lower than expected usage can be addressed by discussing, or requiring, higher frequency of use with the team.
- Ease Of Use - “This practice is easy to use” - Lower than expected Ease of Use can be addressed through, for example, examination and refactoring of work practices, and through training.
- Utility - “This practice assists in preventing or removing security vulnerabilities in our product” - Lower than expected Utility can be addressed through examining practice use, and, possibly, discontinuing use of the practice.

¹<https://www.qualtrics.com/>

- Training - “I have been trained in the use of this practice” - Low training for a practice can be addressed through increasing the availability of training.

We worked with the NCSU Institutional Review Board² to address anonymity and data use concerns in the questionnaire and the participant consent form.

We piloted the survey instrument with NC State researchers and IBM practitioners, incorporating feedback on the wording of the practice definitions and survey questions.

²<https://research.ncsu.edu/sparcs/compliance/irb/>, protocol 6198

Appendix C

SP-EF Guidebook

C.1 SP-EF Measurement Guidebook

C.1.1 Abstract

Software development teams are increasingly faced with security concerns regarding the software they develop. While many software development security practices have been proposed, published empirical evidence for their suitability and effectiveness is limited. The goal of this research is to support theory construction through empirical evidence collection for security practice use in software development by building a measurement framework for software development security practices and their correlations with security-related outcomes.

To orient the framework, we set two practice-oriented sub-goals:

- Identify security practices most likely to reduce post-release vulnerabilities from the pool of practices not currently applied on a project.
- Reduce post-release vulnerabilities for a project through security practice selection and use.

To meet our goals, we define and evaluate the “Security Practices Evaluation Framework” (SP-EF).

This document describes how to collect the data required for SP-EF. Examining patterns in the aggregated data supports making security-focused improvements to the development process.

C.2 Introduction

Vulnerability prevention and removal are becoming increasingly important in software development. A number of security practices have been developed, advocated and implemented, but there is not yet empirical evidence for which practices work best in a given development environment.

In this document, we describe the data elements of the Security Practices Evaluation Framework (SP-EF), and provide guidance on how to collect the data elements. SP-EF contains three categories of data elements; practice adherence metrics, outcome measures, and context factors. The practice adherence metrics are a set of attributes and values that are used to describe security practices in use on a project, and the degree to which each practice is adhered to by the project team. Outcome measures are a set of attributes and values that are used to describe the security-related outcomes of the project. Context factors are a set of attributes and values that are used to provide a basis of comparison between projects measured using SP-EF.

The goal of the SP-EF is to provide a repeatable set of measures and measurement instructions structuring case studies of security practice use so that the case studies can be combined, compared, and analyzed to form a family of evidence on security practice use.^[1]

We have adopted the design of the <http://collaboration.csc.ncsu.edu/laurie/Papers/ease.pdf> Extreme Programming Evaluation Framework (XP-EF), where possible. Goals for the framework include that the metrics be:

- Simple enough for a small team to measure without a metrics specialist and with minimal burden
- Concrete and unambiguous
- Comprehensive and complete enough to cover vital factors

The framework is designed for use throughout development, as well as for annotation of projects that have been completed.

C.2.0.1 Data Sources

The primary data sources required are the project's documentation, particularly all process-related documentation, the version control system, and the bug tracker.

C.2.0.2 Project Measurement Demographics

To identify the collected data, record the following items:

- Organization name - Security practices, personnel policies, media and public attention, and many other factors will vary from organization to organization. We record the organization name to permit controlling for the organization.
- Project name - Development platforms, schedules, staffing Security practices, personnel policies, and many other factors will vary from project to project. We record the project name to permit controlling for the project.
- Date(s) measurements were taken
- Start date of measurement period
- End date of measurement period
- Links or notes on project documentation
- Version control system
- Bug tracking system

C.2.1 Domain

C.2.1.1 Description

Different risks are associated with different software domains. Web applications may be concerned with sustaining thousands, or possibly millions, of concurrent users supporting a variety of different languages. Whereas the primary concerns of a database project may be scalability and response time. The medical domain has unique security and/or privacy concerns.

C.2.1.2 Data Collection

Text description based on discussion with project staff, or read from project artifacts.

C.2.2 Context Factors

Drawing general conclusions from empirical studies in software engineering is difficult because the results of any process largely depend upon the specifics of the study and relevant context factors. We cannot assume a priori that a study's results generalize beyond the specific environment in which it was conducted [3]. Therefore, recording an experiment's context factors is essential for comparison purposes and for fully understanding the similarities and differences between the case study and one's own environment.

C.2.3 Language

C.2.3.1 Description

Language in which the software is written

C.2.3.2 Data Collection

Text description from project staff, researcher observation, or inferred from project artifacts.

C.2.4 Confidentiality, Integrity, and Availability Requirements

C.2.4.1 Description

These values are taken directly from CVSS, and this section paraphrases the description in the <https://www.first.org/cvss/specification-document> CVSS Guide. These metrics measure the security requirements of the software under development. Each security requirement has three possible values: Low, Medium, High, and Not Defined.

C.2.4.2 Data Collection

To choose a value for each context factor, consider the most sensitive data that passes through, or is kept by, the software being evaluated. For example, a web browser may access highly confidential personal information such as bank account or medical record data, to which a High Confidentiality Requirement would apply.

Metric Value Description

- Low (L) Loss of [confidentiality | integrity | availability] is likely to have only a limited adverse effect on the organization or individuals associated with the organization (e.g., employees, customers).
 - Medium (M) Loss of [confidentiality | integrity | availability] is likely to have a serious adverse effect on the organization or individuals associated with the organization (e.g., employees, customers).
 - High (H) Loss of [confidentiality | integrity | availability] is likely to have a catastrophic adverse effect on the organization or individuals associated with the organization (e.g., employees, customers).
 - Not Defined (ND) Assigning this value to the metric will not influence the score. It is a signal to the equation to skip this metric.
-
-

C.2.5 Methodology

C.2.5.1 Description

Project approach to the software development lifecycle.

C.2.5.2 Data Collection

Text description from project staff or researcher observation (e.g., XP, Scrum, Waterfall, Spiral).

C.2.6 Source Code Availability

C.2.6.1 Description

Ross Anderson ¹ has claimed that, for sufficiently large software systems, source code availability aids attackers and defenders equally, but the balance shifts based on a variety of project-specific factors. We track the source code availability for each project measured.

C.2.6.2 Data Collection

Discuss source code availability with the project staff, or infer from the existence of a public repository or other, legal, public distribution of the source code.

Values: Open Source, Closed Source

C.2.7 Team Size

C.2.7.1 Description

The complexity of team management grows as team size increases. Communication between team members and the integration of concurrently developed software becomes more difficult for large teams, as described by https://en.wikipedia.org/wiki/The_Mythical_Man-Month Brooks. Small teams, relative to the size of the project, may be resource-constrained. Therefore, we track the number of people engaged in software development for the project, categorized by project role. To enable normalizing effort and calculation of productivity, we record average hours per week for each person in their primary role.

The four roles defined for SP-EF are:

- Manager (e.gr. Project Management, Requirements Engineer, Documentation, Build Administrator, Security),
- Developer (Designer, Developer),
- Tester (Quality Assurance, Penetration Tester, External Penetration Tester),
- Operator (User, Systems Administrator, Database Administrator),

¹<https://www.cl.cam.ac.uk/~rja14/Papers/toulouse.pdf>

C.2.7.2 Data Collection

Count managers, developers, and testers dedicated to the project under study.

Survey project team to establish each member's time commitment to the project.

Count When working with a project in progress, count people currently engaged on the project, noting roles and whether they are full-time or part-time on the project. When working with historical project data, sort participants by their number of commits (or bug reports) and count participants contributing the first 80% of commits (bug reports) to estimate development team size and testing team size.

Per team member data:

- Project Role Values: Manager, Developer, Tester, Other.
- Average Hours Per Week: 0.0 - 99.9

Team size: Summary by Project Role, Count, Average Hours Per Week

C.2.8 Team Location

C.2.8.1 Description

Distributed teams that communicate via the Internet are becoming more commonplace, and it is possible that team location and accessibility may influence a project. A distributed team faces more challenges than a co-located team during development. Communication and feedback times are typically increased when the team is distributed over many sites.

C.2.8.2 Data Collection

Record whether the team is colocated or distributed. A colocated team is found in the same building and area, such that personal interaction is easily facilitated. If the team is distributed, record whether the distribution is across several buildings, cities, countries, or time zones.

C.2.9 Operating System

C.2.9.1 Description

Operating system/runtime environment and version.

C.2.9.2 Data Collection

Text description from project staff or researcher observation (e.g. Linux, Windows, Android, iOS).

C.2.10 Product Age

C.2.10.1 Description

Product age relates to both the availability of product knowledge as well as product refinement. An older product might be considered more stable with fewer defects, but there may be a lack of personnel or technology to support the system. Furthermore, making significant changes to a legacy system may be an extremely complex and laborious task. Working with a newer product may involve instituting complex elements of architectural design that may influence subsequent development, and may be prone to more defects since the product has not received extensive field use.

C.2.10.2 Data Collection

Determine the date of the first commit/first lines of code written. Record the number of months elapsed since that date. Record the age of the product in months.

C.2.11 Number of Identities

C.2.11.1 Description

Number of personal identities the software stores or transmits.

A black market for personal identities, names, addresses, credit card numbers, bank account numbers, has developed. In 2011, a personal identity could be bought (in groups of 1000) for 16 US cents^[3]. One component of software security risk is the presence and use of personal information, represented by the number of identities accessible to the software.

C.2.11.2 Data Collection

Work with the team to count or estimate the number of personal identities managed by the software. A browser might manage one or two identities, while a database system might manage millions.

C.2.12 Number of Machines

C.2.12.1 Description

Number of machines on which the software runs.

The rise of botnets, networks of computers that can be centrally directed, has created a black market for their services. In 2013, an hour of machine time on a botnet ranged from 2.5 to 12 US cents^{ref}. Infesting machines with malware enabling central control creates Botnets, and so the number of machines a piece of software runs on is a risk factor.

C.2.12.2 Data Collection

Work with team to count or estimate the machines (physical or virtual) on which the software runs.

C.2.13 Source Lines of Code (SLOC)

“Measuring programming progress by lines of code is like measuring aircraft building progress by weight.” - Bill Gates

C.2.13.1 Description

Lines of Code is one of the oldest, and most controversial, software metrics. We use it as a means for assessing software size, and as a proxy for more detailed measures such as complexity. Broadly speaking, larger code size may indicate the potential for software defects, including vulnerabilities.

C.2.13.2 Definition

Number of non-blank, non-comment lines present in the release of the software being working on during the current project.

C.2.13.3 Data Collection

Count total number of non-blank, non-comment lines present in the release of the software being working on during the current project.

Use <https://github.com/AlDanial/cloc> or <http://www.dwheeler.com/sloccount/> SLOCCount where possible.

C.2.14 Churn

C.2.14.1 Description

Development teams change software to add features, to correct defects, and to refine software performance according to variety of non-functional requirements. Changes to software can introduce defects, and so measuring change is important for assessing software quality. We measure *Churn*, the number of non-blank, non-comment lines changed, added, or deleted in the the software being working on, over a time period. Churn is composed of three measurements; Start Date, End Date, and the total changed, added, deleted SLOC between the Start Date and the End Date.

C.2.14.2 Data Collection

Select the Start Date and End Date to be measured. In our initial studies, we define Project Month, and compute Churn for each month since the first available month of data for the project, using the first and last days of each month as our Start and End Dates. In other studies, Start and End Dates may be for a single release, or series of releases.

Following the data collection procedures for SLOC, measure SLOC for the Start Date. Measure changed, added, deleted SLOC for the End Date, relative to the Start Date.

SP-EF's practice adherence metrics are designed to measure how closely a project adheres to a set of security practices. Each project is likely to use its own set of security practices and to use a given security practice in its own way. Project teams may add and drop practices to suit the requirements of their customers, their business and operational environments, and their awareness of trends in software development. Adherence metrics are a means of characterizing the degree to which a practice is used on a project. We have included objective and subjective metrics for measuring practice adherence. Objective metrics are drawn from evaluation of the project data, given our expectation that the security practices of a team will be reflected in the documentation the team creates, and the logs of activity the team generates.

Subjective metrics are drawn from interviews with, or surveys of, the project team members. People are the driving force behind process and practices, and their views should be considered, while weighing the bias introduced by self-reporting.

For each security practice adherence event, we recorded the following data elements: Event Date – Date on which document was created. * Frequency: Frequency with which the practice is performed. Values: Not Used, Daily, Weekly, Monthly, Quarterly, Annually, Less than Annually Practice – Name of security practice associated with document. Source – Data source for document. Possible Values: Version Control, Defect Tracker, Email. Document Id – Id of document in its source, e.g. commit hash, bug tracker id, email id. Creator – Role of the author of the source document. Role: Manager, Developer, Tester, User, Attacker. Assignee – For defect report documents, the person assigned the defect, where applicable * Phase - Project phase during which the practice is performed. Values: Initiation, Requirements, Design, Implementation, Unit Testing, Testing, Release, Operations * Evidence source: link to, or description of, source of data reported.

While the practice adherence metrics are, conceivably, useful for any set of practices, we have defined a set of software development security practices, synthesized from our evaluation

of the BSIMM, SDL, SAFECode, and OWASP practice lists.

C.2.15 Perform Security Training

Ensure project staff are trained in security concepts, and in role-specific security techniques.

C.2.16 Description

Security training raises staff awareness of potential security risks and approaches for mitigating those risks. While some security concepts, e.g. Confidentiality, Availability, and Integrity, apply in general, role-specific training, e.g. coding techniques, database management, design concerns, is beneficial.

C.2.16.1 Practice Implementation Questions

1. Is security treated as part of the on boarding process?
2. Are project staff trained in the general security principles?
3. Are project staff trained in the security techniques they are expected to apply?
4. Is refresher training scheduled periodically?
5. Is further training scheduled periodically?
6. Are security mentors available to the project?

C.2.16.2 Keywords

awareness program, class, conference, course, curriculum, education, hiring, refresher, mentor, new developer, new hire, on boarding, teacher, training

C.2.16.3 Links

C.2.17 Apply Data Classification Scheme

Maintain and apply a Data Classification Scheme. Identify and document security-sensitive data, personal information, financial information, system credentials.

C.2.17.1 Description

A Data Classification Scheme (DCS) specifies the characteristics of security-sensitive data, for example, personal information, financial information, and/or system credentials. The DCS should be developed by considering the security implications of all data used by the software. The DCS should be considered by project personnel when writing, testing, and documenting the project's software.

C.2.17.2 Practice Implementation Questions

1. Does the software under development reference, store, or transmit any of the following data:
 - personally-identifiable information (PII)
 - financial information
 - credit card information
 - system credentials (e.g. passwords, ssh keys)
2. Are rules for recognizing all of the data types used in question 1 documented?
3. Are rules for handling all of the data types used in question 1 documented?
4. Is the DCS revised periodically?
5. Are all personnel trained in the use of the DCS?
6. Are personnel periodically re-trained in the use of the DCS?

C.2.17.3 Keywords

(street) address, credit card number, data classification, data inventory, Personally Identifiable Information (PII), user data, privacy.

C.2.18 Apply Security Requirements

Consider and document security concerns prior to implementation of software features.

C.2.18.1 Description

Security requirements are documented statements about what the software should allow and ban with respect to security concerns, including confidentiality, integrity, and availability. When

a developer (tester) works on a piece of code (test), they should be able to reference the security requirements of that code (test).

C.2.18.2 Practice Implementation Questions

1. Are there organizational and/or project standards for documenting security requirements?
2. Is a plan for how security will be addressed during development created before development begins?
3. Does the software development team know whether compliance (regulatory, and organizational standards) requirements apply to its software?
4. Are compliance requirements translated into the work items/user stories/functional specs the developers use to guide their day to day progress?
5. Are user roles, behavior, and permissions specified before coding?
6. Are the environments and corresponding trust boundaries under which the software will run considered during design/before coding?
7. Are authentication and authorization implemented for the services and data the software provides?

C.2.18.3 Keywords

authentication, authorization, requirement, use case, scenario, specification, confidentiality, availability, integrity, non-repudiation, user role, regulations, contractual agreements, obligations, risk assessment, FFIEC, GLBA, OCC, PCI DSS, SOX, HIPAA.

C.2.19 Apply Threat Modeling

Anticipate, analyze, and document how and why attackers may attempt to misuse the software.

C.2.19.1 Description

Threat modeling is the process of analyzing how and why attackers might subvert security mechanisms to gain access to the data and other assets accessible through the project's software.

C.2.19.2 Practice Implementation Questions

1. Does the project have a standard for threat modeling?

2. Does the project have a list of expected attackers?
3. Does the project have a list of expected attacks?
4. Does the project budget for time to analyze its expected attackers and attacks, identify vulnerabilities, and plan for their resolution?
5. Does the project budget time for keeping up to date on new attackers and attacks?
 - for the project software
 - for the project technologies
 - for the environment in which the project operates?
6. Does the project develop ‘abuse cases’ or ‘misuse cases’ based on its expected attackers?
7. Are defect records created to track resolution of each vulnerability discovered during threat modeling?
8. Are results from vulnerability tracking fed into the threat modeling process?

C.2.19.3 Keywords

threats, attackers, attacks, attack pattern, attack surface, vulnerability, exploit, misuse case, abuse case.

C.2.19.4 Links

https://www.owasp.org/index.php/Threat_modeling
OWASP Threat Modeling

C.2.20 Document Technical Stack

Document the components used to build, test, deploy, and operate the software. Keep components up to date on security patches.

C.2.20.1 Description

The technical stack consists of all software components required to operate the project’s software in production, as well as the software components used to build, test, and deploy the software. Documentation of the technical stack is necessary for threat modeling, for defining a repeatable development process, and for maintenance of the software’s environment when components receive security patches.

C.2.20.2 Practice Implementation Questions

1. Does the project maintain a list of the technologies it uses?
2. Are all languages, libraries, tools, and infrastructure components used during development, testing, and production on the list?
3. Are security features developed by the project/organization included on the list?
4. Is there a security vetting process required before a component is added to the list?
5. Is there a security vetting process required before a component is used by the project?
6. Does the list enumerate banned components?
7. Does the project review the list, and vulnerabilities of components on the list? On a schedule?

C.2.20.3 Keywords

stack, operating system, database, application server, runtime environment, language, library, component, patch, framework, sandbox, environment, network, tool, compiler, service, version

C.2.20.4 Links

<https://www.bsimm.com/online/intelligence/sr/?s=sr2.3#sr2.3\BSIMM R2.3> Create standards for technology stacks.

C.2.21 Apply Secure Coding Standards

Apply (and define, if necessary) security-focused coding standards for each language and component used in building the software.

C.2.21.1 Description

A secure coding standard consists of security-specific usage rules for the language(s) used to develop the project's software.

C.2.21.2 Practice Implementation Questions

1. Is there a coding standard used by the project?
2. Are security-specific rules included in the project's coding standard?
3. Is logging required by the coding standard?

4. Are rules for cryptography (encryption and decryption) specified in the coding standard?
5. Are technology-specific security rules included in the project's coding standard?
6. Are good and bad examples of security coding given in the standard?
7. Are checks of the project coding standards automated?
8. Are project coding standards enforced?
9. Are project coding standards revised as needed? On a schedule?

C.2.21.3 Keywords

avoid, banned, buffer overflow, checklist, code, code review, code review checklist, coding technique, commit checklist, dependency, design pattern, do not use, enforce function, firewall, grant, input validation, integer overflow, logging, memory allocation, methodology, policy, port, security features, security principle, session, software quality, source code, standard, string concatenation, string handling function, SQL Injection, unsafe functions, validate, XML parser

C.2.21.4 Links

<https://www.bsimm.com/online/intelligence/sr/?s=sr1.4#sr1.4>

BSIMM SR 1.4 Use secure coding standards

C.2.22 Apply Security Tooling

Use security-focused verification tool support (e.g. static analysis, dynamic analysis, coverage analysis) during development and testing.

C.2.22.1 Description

Use security-focused verification tool support (e.g. static analysis, dynamic analysis, coverage analysis) during development and testing. Static analysis tools apply verification rules to program source code. Dynamic analysis tools apply verification rules to running programs. Fuzz testing is a security-specific form of dynamic analysis, focused on generating program inputs that can cause program crashes. Coverage analyzers report on how much code is 'covered' by the execution of a set of tests. Combinations of static, dynamic, and coverage analysis tools support verification of software.

C.2.22.2 Practice Implementation Questions

1. Are security tools used by the project?
2. Are coverage analyzers used?
3. Are static analysis tools used?
4. Are dynamic analysis tools used?
5. Are fuzzers used on components that accept data from untrusted sources (e.g. users, networks)?
6. Are defects created for (true positive) warnings issued by the security tools?
7. Are security tools incorporated into the release build process?
8. Are security tools incorporated into the developer build process?

C.2.22.3 Keywords

automate, automated, automating, code analysis, coverage analysis, dynamic analysis, false positive, fuzz test, fuzzer, fuzzing, malicious code detection, scanner, static analysis, tool

C.2.22.4 Links

<https://www.bsimm.com/online/ssdl/cr/?s=cr1.4#cr1.4>

Use automated tools along with manual review.

C.2.23 Perform Security Review

Perform security-focused review of all deliverables, including, for example, design, source code, software release, and documentation. Include reviewers who did not produce the deliverable being reviewed.

C.2.23.1 Description

Manual review of software development deliverables augments software testing and tool verification. During review, the team applies its domain knowledge, expertise, and creativity explicitly to verification rather than implementation. Non-author reviewers, e.g. teammates, reviewers from outside the team, or security experts, may catch otherwise overlooked security issues.

C.2.23.2 Practice Implementation Questions

Each of the following questions applies to the decision to:

- change code, configuration, or documentation
 - include a (revised) component the project
 - release the (revised) software built by the project
1. Does the project use a scheme for identifying and ranking security-critical components?
 2. Is the scheme used to prioritize review of components?
 3. Are the project's standards documents considered when making the decision?
 4. Are the project's technical stack requirements considered when making the decision?
 5. Are the project's security requirements considered when making the decision?
 6. Are the project's threat models considered when making the decision?
 7. Are the project's security test results considered when making the decision?
 8. Are the project's security tool outputs considered when making the decision?
 9. Are changes to the project's documentation considered when making the decision?

C.2.23.3 Keywords

architecture analysis, attack surface, bug bar, code review, denial of service, design review, elevation of privilege, information disclosure, quality gate, release gate, repudiation, review, security design review, security risk assessment, spoofing, tampering, STRIDE

C.2.23.4 Links

https://www.owasp.org/index.php/Perform_source-level_security_review

OWASP Perform source-level security review

<https://www.bsimm.com/online/ssdl/cr/?s=cr1.5#cr1.5>

BSIMM CR1.5 Make code review mandatory for all projects.

C.2.24 Perform Security Testing

Consider security requirements, threat models, and all other available security-related information and tooling when designing and executing the software's test plan.

C.2.25 Description

Testing includes using the system from an attacker's point of view. Consider security requirements, threat model(s), and all other available security-related information and tooling when developing tests. Where possible, automate test suites, and include security-focused tools.

C.2.25.1 Practice Implementation Questions

1. Is the project threat model used when creating the test plan?
2. Are the project's security requirements used when creating the test plan?
3. Are features of the technical stack used by the software considered when creating the test plan?
4. Are appropriate fuzzing tools applied to components accepting untrusted data as part of the test plan?
5. Are tests created for vulnerabilities identified in the software?
6. Are the project's technical stack rules checked by the test plan?
7. Is the test plan automated where possible?
8. Are the project's technical stack rules enforced during testing?

C.2.25.2 Keywords

boundary value, boundary condition, edge case, entry point, input validation, interface, output validation, replay testing, security tests, test, tests, test plan, test suite, validate input, validation testing, regression test

C.2.26 Links

https://www.owasp.org/index.php/Identify,_implement,_and_perform_security_tests
OWASP Identify, implement, and perform security tests, <https://www.bsimm.com/online/ssdl/st/?s=st3.2#st3.2BSIMM-ST3.2> Perform fuzz testing customized to application APIs

C.2.27 Publish Operations Guide

Document security concerns applicable to administrators and users, supporting how they configure and operate the software.

C.2.27.1 Description

The software's users and administrators need to understand the security risks of the software and how those risks change depending on how the software is configured. Document security concerns applicable to users and administrators, supporting how they operate and configure the software. The software's security requirements and threat model are expressed in the vocabulary of the user (and administrator).

C.2.27.2 Practice Implementation Questions

1. Are security-related aspects of installing and configuring the software documented where users can access them?
2. Are security-related aspects of operating the software documented where users can access them?
3. Are abuse cases and misuse cases used to support user documentation?
4. Are expected security-related alerts, warnings and error messages documented for the user?

C.2.27.3 Keywords

administrator, alert, configuration, deployment, error message, guidance, installation guide, misuse case, operational security guide, operator, security documentation, user, warning

C.2.27.4 Links

https://www.owasp.org/index.php/Build_operational_security_guide
OWASP Build operational security guide

C.2.28 Perform Penetration Testing

Arrange for security-focused stress testing of the project's software in its production environment. Engage testers from outside the software's project team.

C.2.28.1 Description

Testing typically is focused on software before it is released. Penetration testing focuses on testing software in its production environment. Arrange for security-focused stress testing of

the project's software in its production environment. To the degree possible, engage testers from outside the software's project team, and from outside the software project's organization.

C.2.28.2 Practice Implementation Questions

1. Does the project do its own penetration testing, using the tools used by penetration testers and attackers?
2. Does the project work with penetration testers external to the project?
3. Does the project provide all project data to the external penetration testers?
4. Is penetration testing performed before releases of the software?
5. Are vulnerabilities found during penetration test logged as defects?

C.2.28.3 Keywords

penetration

C.2.28.4 Links

https://www.owasp.org/index.php/Web_Application_Penetration_Testing
OWASP Web Application Penetration Testing

C.2.29 Track Vulnerabilities

Track software vulnerabilities detected in the software, and prioritize their resolution.

C.2.29.1 Description

Vulnerabilities, whether they are found in development, testing, or production, are identified in a way that allows the project team to understand, resolve, and test quickly and efficiently. Track software vulnerabilities detected in the software, and prioritize their resolution.

C.2.29.2 Practice Implementation Questions

1. Does the project have a plan for responding to security issues (vulnerabilities)?
2. Does the project have an identified contact for handling vulnerability reports?
3. Does the project have a defect tracking system?
4. Are vulnerabilities flagged as such in the project's defect tracking system?

5. Are vulnerabilities assigned a severity/priority?
6. Are vulnerabilities found during operations recorded in the defect tracking system?
7. Are vulnerabilities tracked through their repair and the re-release of the affected software?
8. Does the project have a list of the vulnerabilities most likely to occur, based on its security requirements, threat modeling, technical stack, and defect tracking history?

C.2.29.3 Keywords

bug, bug bounty, bug database, bug tracker, defect, defect tracking, incident, incident response, severity, top bug list, vulnerability, vulnerability tracking

C.2.29.4 Links

<https://www.bsimm.com/online/deployment/cmvm/?s=cmvm2.2#cmvm2.2>

BSIMM CMVM 2.2 Track software bugs found during ops through the fix process.

C.2.30 Improve Development Process

Incorporate “lessons learned” from security vulnerabilities and their resolutions into the project’s software development process.

C.2.30.1 Description

Experience with identifying and resolving vulnerabilities, and testing their fixes, can be fed back into the development process to avoid similar issues in the future. Incorporate “lessons learned” from security vulnerabilities and their resolutions into the project’s software development process.

C.2.30.2 Practice Implementation Questions

1. Does the project have a documented standard for its development process?
2. When vulnerabilities occur, is considering changes to the development process part of the vulnerability resolution?
3. Are guidelines for implementing the other SPEF practices part of the documented development process?
4. Is the process reviewed for opportunities to automate or streamline tasks?
5. Is the documented development process enforced?

C.2.30.3 Keywords

architecture analysis, code review, design review, development phase, gate, root cause analysis, software development lifecycle, software process

C.2.30.4 Links

<https://www.bsimm.com/online/governance/cp/?s=cp3.3#cp3.3>

BSIMM CP 3.3 Drive feedback from SSDL data back to policy.

C.2.31 Subjective Practice Adherence Measurement

Text-based practice adherence data collection.

C.2.31.1 Description

SP-EF includes five subjective adherence measures that can be used in surveys and interviews:

- Usage - How often is this practice applied?
 - Values: not used, daily, weekly, monthly, quarterly, annually, less than annually.
 - Ease Of Use - How easy is this practice to use?
 - Values: Very Low, Low, Nominal, High, Very High.
 - Utility - How much does this practice assist in providing security in the software under development?
 - Values: Very Low, Low, Nominal, High, Very High.
 - Training - How well trained is the project staff in the practices being used?
 - Values: Very Low, Low, Nominal, High, Very High.
 - Effort - How much time, on average, does applying this practice take each time you apply it?
 - Ordinal values: 5 minutes or less, 5-15 minutes, 15-30 minutes, 30-minutes-1 hour, 1-4 hours, 4-8 hours, 1-2 days, 3-5 days, over 5 days
 - Ratio values: hours (fractional allowed)
-

C.2.32 Objective Practice Adherence Measurement

Practice adherence data based on concrete project data.

C.2.32.1 Description

Objective metrics are drawn from evaluation of the project data, given our expectation that the security practices of a team will be reflected in the documentation the team creates, and the logs of activity the team generates.

We collect the following objective practice adherence metrics for each practice:

- Presence: whether we can find evidence of the practice.
 - Values: True, False.
- Prevalence: Proportion of the team applying the practice, the ratio of all practice users to all team members.
 - Values: 0 - 1.00.
 - Alternate Values: Low, Medium, High.

When recording practice adherence manually, it is sufficient to record the following data elements:

- Practice - Name of security practice associated with document.
- Practice date: Date for which evidence of practice use is claimed by the researcher.
- Presence - as described above
- Prevalence - as described above

When recording practice adherence events automatically from emails, issues, commits, we recorded the following data elements:

- Practice - Name of security practice associated with document.
- Event Date - Date on which document was created.
- Source - Data source for document. Possible Values: Version Control, Defect Tracker, Email.
- Document Id - Id of document in its source, e.g. commit hash, bug tracker id, email id.
- Creator - Role of the author of the source document.
- Assignee - For defect report documents, the person assigned the defect, where applicable.

C.2.33 Per Vulnerability Attributes

C.2.33.1 Description

While hundreds of security metrics have been proposed, tracking a relatively small set of attributes for each vulnerability detected in the software is sufficient to replicate many of them.

C.2.33.2 Definition

C.2.33.3 Data Collection

In addition to data kept for defects (e.g. those attributes listed by Lamkanfi [41]), we collect:

- Source – The name of the bug tracker or bug-tracking database where the vulnerability is recorded.
- Identifier – The unique identifier of the vulnerability in its source database.
- Description – Text description of the vulnerability.
- Discovery Date – Date the vulnerability was discovered.
- Creation Date – Date the tracking record was created.
- Patch Date – The date the change resolving the vulnerability was made.
- Release Date – The date the software containing the vulnerability was released.
- Severity – The criticality of the vulnerability. Scale: Low, Medium, High.
- Phase – Indication of when during the development lifecycle the vulnerability was discovered
- Reporter – Indication of who found the vulnerability
- Role
- (Optional) Identifier (name, email)

C.2.34 Pre-Release Defects

C.2.34.1 Description

Defects discovered during the development process should be credited to the team and its development practices.

C.2.34.2 Definition

Defects found in new and changed code before software is released.

C.2.34.3 Data Collection

When a defect is found in new or changed code before the software is released, collect the Per-Defect attributes and mark the development phase where the software was found; Requirements, Design, Development, Testing. Count total number of Defects found in new and changed code before the software is released.

C.2.35 Post-Release Defects

C.2.35.1 Description

Defects discovered after the software is released should be studied for how they could be identified and resolved sooner.

C.2.35.2 Definition

Defects found in released software.

C.2.35.3 Data Collection

When a vulnerability is found in released software, record its per-vulnerability attributes and mark the Phase as 'Post-Release'. Count total number of Defects found in released software.

C.2.36 Vulnerability Density

C.2.36.1 Description

Vulnerability Density (Vdensity) is the cumulative vulnerability count per unit size of code. We adopt a size unit of thousand source lines of code (KSLOC).

C.2.36.2 Definition

Total Vulnerabilities divided by number of KSLOC in the software, at a point in time.

C.2.36.3 Data Collection

Derived from Pre- and Post-Release Vulnerabilities and SLOC metrics.

C.2.37 Pre-Release Vulnerabilities

C.2.37.1 Description

Vulnerabilities discovered during the development process should be credited to the team and its development practices.

C.2.37.2 Definition

Vulnerabilities found in new and changed code before software is released.

C.2.37.3 Data Collection

When a vulnerability is found in new or changed code before the software is released, Collect the Per-Vulnerability attributes and mark the development phase where the software was found; Requirements, Design, Development, Testing. Count total number of vulnerabilities found in new and changed code before the software is released.

C.2.38 Post-Release Vulnerabilities

C.2.38.1 Description

Vulnerabilities discovered after the software is released should be studied for how they could be identified and resolved sooner.

C.2.38.2 Definition

Vulnerabilities found in released software.

C.2.38.3 Data Collection

When a vulnerability is found in released software, record its per-vulnerability attributes and mark the Phase as 'Post-Release'. Count total number of vulnerabilities found in released software.

C.2.39 Vulnerability Removal Effectiveness

C.2.39.1 Description

Vulnerability Removal Effectiveness (VRE) is the ratio of pre-release vulnerabilities to total vulnerabilities found, pre- and post-release, analogous to defect removal effectiveness. Ideally, a development team will find all vulnerabilities before the software is shipped. VRE is a measure for how effective the team's security practices are at finding vulnerabilities before release.

C.2.39.2 Definition

Pre-Release Vulnerabilities divided by total number of Pre- and Post-Release Vulnerabilities in the software, at a point in time.

C.2.40 Data Collection

Derived from Pre- and Post-Release Vulnerabilities metrics.

@ Patrick Morrison 2015 | All Rights Reserved.