FINAL TECHNICAL REPORT SERC-2021-TR-009

WRT 1016
REDUCING TOTAL OWNERSHIP COST (TOC) AND SCHEDULE

Date: JANUARY 17, 2021

PRINCIPAL INVESTIGATOR: DR. BARRY BOEHM, UNIVERSITY OF SOUTHERN CALIFORNIA

Sponsor(s): Office of the Under Secretary of Defense for Research & Engineering

DISCLAIMER

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract [HQ0034-19-D-0003, TO#0620].

RESEARCH TEAM

Name	Org.	Labor Category
Barry Boehm	USC	Principal Investigator
Pooyan Behnamghader	USC	Post Doctorate Researcher
Michael Shoga	USC	Graduate Research Assistant (PhD)
Iordanis Fostiropoulos	USC	Graduate Research Assistant (PhD)
Jincheng He	USC	Graduate Research Assistant (PhD)
Elaine Venson	USC	Graduate Research Assistant (PhD)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF (TABLES, SEQUENCES)

EXECUTIVE SUMMARY

An essential driver for reducing Total Ownership Cost (TOC) and schedule is maintainability. This system quality (SQ) is key to reducing 75% of most systems' life cycle costs. Also, Maintainability plays a key role in other top-level SQs: Life Cycle Efficiency, Dependability, and Changeability. Dependability needs Maintainability to relate Reliability to Availability; and Changeability needs Maintainability to address new system challenges and opportunities.

USC developed a tool called the Software Qualities Understanding by Analysis of Abundant Data (SQUAAD) for use in analysis of software technical debt[1]. SQUAAD has recently been extended to identify additional sources of technical debt by using *uncompilability* (computer code that fails to convert into machine instructions) as a symptom of careless development and analyzing software quality evolution over sequences of uncompilable commits. A commit is an event where computer code is modified, returned to the software code repository, and compiled (converted into machine instructions).

Another research focus was on the categorization for software commits and investigating how multiple-categories in a commit impacts software quality. Uncompilability increased when a software commit had more than two categories of change indicating a reduction in software quality. Work continues in refining categorization and training models to automate categorizing commits.

Research was also done on the identification of synergies and conflicts between system qualities. Using computer code from open-source Apache projects and the CAST tool, a correlation analysis was done to detect synergistic and conflicting qualities across ten different quality metrics. This research can provide a better understanding of what trade-offs will need to be made and the costs associated with ensuring levels of different qualities for the intended system.

A new security cost driver has been developed for COCOMO II and for later use in the emerging definition of COCOMO III. This involved surveys of the Linked-In security community, presentation and discussion of the proposed security cost driver rating scale at the SERC Doctoral Forum, and an experts' workshop at the 2019 annual COCOMO Forum, and an experts' Delphi consensus of the cost multipliers for each of the rating scale levels. This work will result in understanding how the total cost of a software system can be reduced with the appropriate allocation of resources in the early stages of the software project, thus reducing TOC.

RESEARCH OVERVIEW

The WRT-1016 project proposed to research and develop systems and software technology that enables future DoD mission-critical systems to more cost-effectively cope with increasingly challenging threats. It proposes to address future challenges in the context of the four SERC Research Council roadmaps. Digital Engineering addresses Life Cycle Maintainability and DoD Systems of Systems Interoperability; Security is extended to include Safety; AI/Autonomy and

[1] Technical debt a concept in software development that reflects the implied cost of additional rework caused by choosing an easy or lower quality solution now instead of using a better approach that would take longer.

Velocity are also directly addressed.

The project also builds on the Systems Quality Ontology developed in SERC RTs 46 through 209, along with anticipatory technology for improving Maintainability during development, such as Maintainability Opportunity Trees, technology for identifying a software system's technical debt such as the Software Qualities Understanding by Analysis of Abundant Data (SQUAAD), and addressing non-technical sources of technical debt. These have led to their experimental use in Navy applications and discussions and demonstrations for FFRDCs such as MITRE for SQUAAD and Aerospace for natural language processing to determine root causes of software problem reports, and our lead participation in the recent SERC workshop on the use of Continuous Development and Deployment (CD&D) on future DoD systems and systems of systems. A recent example of SQUAAD scalability was the analysis of technical debt increases and decreases of 1.3 billion lines of code across 5 to 15 years of software development and maintenance of Google, Apache, and Netflix code. More details on SQUAAD and other methods and tools needed for DoD systems are provided in our recently-published Wiley Systems Engineering journal article [1].

Discussions with MITRE and other potential users indicated that a version of SQUAAD based on a private cloud would be needed for classified work, and user-interface features developed for use by non-specialists. This will be a main task, along with identifying potential early users, supporting their use, and addressing their usage suggestions. Other tasks include research and experimentation in extending SQUAAD to identify additional sources of technical debt or deficiencies in security, safety, autonomy, interoperability, and velocity.

Considerable additional research and experimentation are needed to address the ontology and tools support for assessing deficiencies in security, safety, autonomy, interoperability, and velocity, and to identify synergies and conflicts among the additional system qualities. This were to begin in optional follow-on research.

1.1 Software Qualities Understanding by Analysis of Abundant Data (SQUAAD)

1.1.1 Purpose

Dr. Pooyan Behnamghader and his team carried out research and experimentation in extending SQUAAD to identify additional sources of technical debt by using *uncompilability* (computer code that fails to convert into machine instructions) as a symptom of careless development and analyzing software quality evolution over sequences of uncompilable commits. The results of this research are recently accepted for publication at the 20th IEEE International Conference on Software Quality, Reliability, and Security (QRS) [2] as a regular paper in the research track. This work consists of the following contributions:

1) a methodology for identifying and assessing software quality change over sequences of uncompilable commits,

2) a taxonomy for categorizing commit purpose including different aspects of software maintainability,

3) a dataset of 914 commits tagged using this taxonomy,

4) findings on the relationship between commit size and purpose, and

5) results on the incidence of commit purpose over uncompilable commits.

A joint workshop was held with CAST, Inc. to explore the integration of USC and CAST qualities assessment capabilities. Data has been collected using SQUAAD and CAST tools, including CISQ Common Weakness Evaluations, to be used in publications by Ph.D. and MS students at USC. A half-day tutorial [3] was held at the 24th annual Aerospace Corp. Ground System Architectures Workshop (GSAW) to summarize current research and operational results from performing large-scale data analysis on large-scale software technical debt using SQUAAD as an example. A poster was presented on architecture design for SQUAAD and its applications at the 2019 SSRR [4]. A paper was published on SQUAAD results in the proceedings of the IEEE Empirical Software Engineering and Measurement conference [5]. Analysis was also presented on the latest results conducted by SQUAAD at the 34th Annual Forum on COCOMO and Systems and Software Cost Estimation and USC CSSE 28th Annual Research Review.

1.1.2 ON-GOING WORK

Figure 1 below shows SQUAAD's analysis of the coevolution of SLOC (an indicator of size) and Code Smells (an indicator of technical debt) in the core module of an open-source software system over a period of 9 years. The analysis includes more than 1200 software changes (commits), and all contributions of each developer are denoted by a unique color. None of the two trends increases monotonically over time, however, the size analysis shows less variation. The ratio of commits that increase the size to the ones that decrease size is 3.00. The ratio for the number of code smells is 1.45.

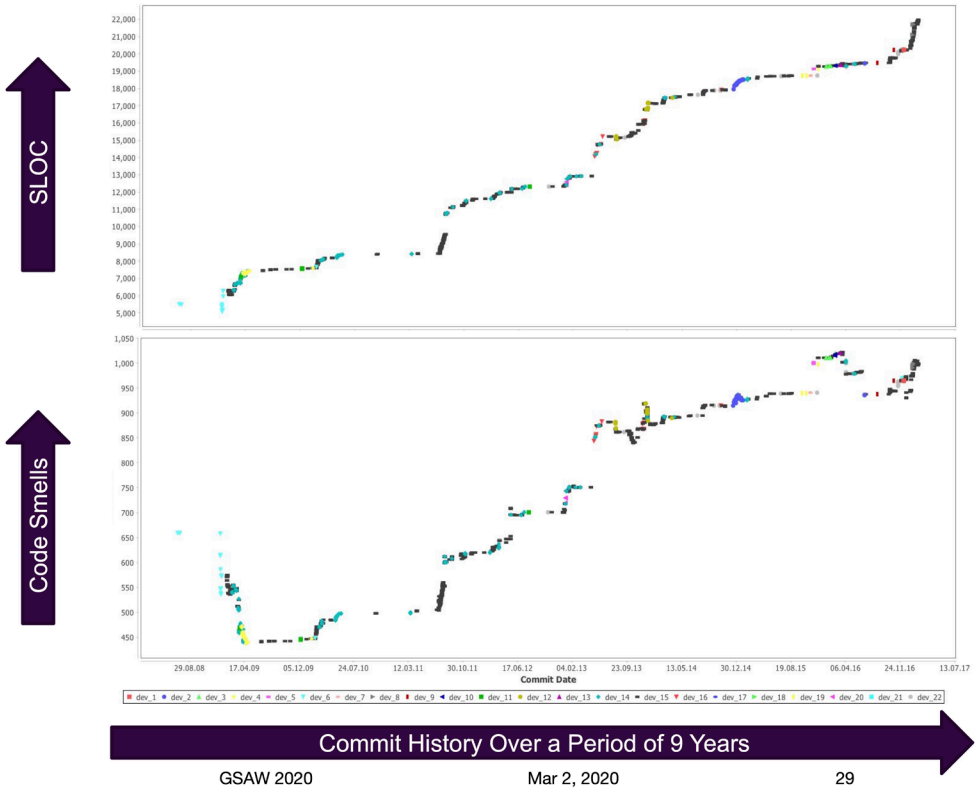

Figure 1. History of Commits Over Time

The SQUAAD tool is being extended to data mine large open-source computer code to investigate quality categorization and software quality interrelationships.

1.2 Categorizational Study on Characteristics of Commits and Their Impacts on Software Quality

1.2.1 Purpose

Mr. Jincheng He's research with respect to WRT-1016 is creating a purpose-oriented categorization for software commits and investigating how different types of commits impact software quality. A commit is an event where computer code is modified, returned to the software code repository, and compiled (converted into machine instructions). The commit categories are Bug Fix, Build, Documentation, Feature Add, Maintenance, and Refactoring. The investigation looks at the number of commit categories in a single commit and assess if the commit broke the software (called a Breaker) thus delaying development and increasing TOC.

Developing software with the source code open to the public is very common; however, similar to its closed counterpart, open source has quality problems, which cause functional failures, such as an unsatisfactory user experience, and non-functional failures, such as long response time. Previous researchers have revealed when, where, how, and what the developers contribute to projects and how these aspects impact software quality. However, there has been little work on how different categories of commits impact software quality. To improve the quality of open-source software, thus reducing total ownership cost, Mr. He is creating a purpose-oriented categorization for software commits and investigating how different types of commits impact software quality. After identifying these impacts, a new set of guidelines will be established for committing changes, thus improving the quality.

1.2.2 ON-GOING WORK

Figure 2 shows one aspect of how multi-purpose commit categories impact software quality (with respect to compilability - computer code that fails to convert into machine instructions). Neutral (compilable) commits are less likely to have multiple purposes while the counterpart, Breaker (uncompilable) have multiple purposes. From this figure, we conclude that multi-purpose commits are more likely to introduce uncompilable commits (grey bars in the figure), thus negatively impacting software quality and lengthening development time.

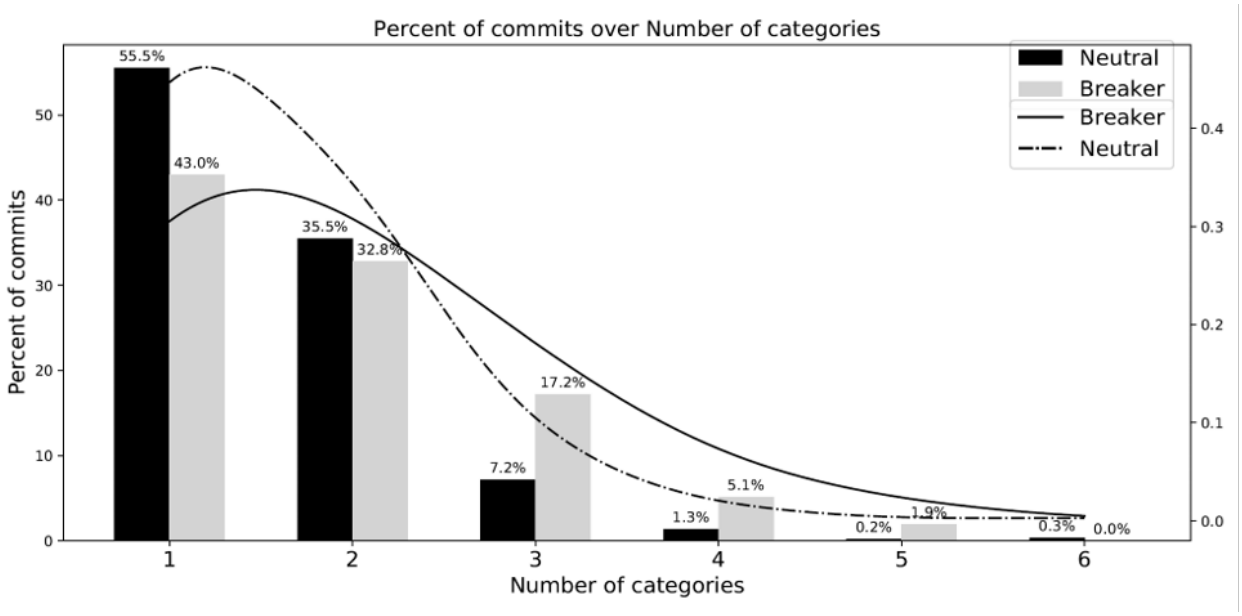

Figure 2. Percent of Commits Over the Number of Categories

After refinement of the categorization, the data is being utilized to train prediction models. For instance, one model is being trained with the purpose of a commit and the commit's meta-data (for example, commit message) to automate tagging commits. Another model is being trained with the purpose of the commit and quality metrics to predict the potential impact on quality.

Additionally, the plan is to further calibrate the commit categorizations based on the code changes to reduce the ambiguity and overlap between categories.

There is also a web application under development to visualize the above works, data, and to run

the analysis.

1.3 PRELIMINARY STUDY ON SOFTWARE QUALITY INTERRELATIONSHIPS IN OPEN-SOURCE SOFTWARE

1.3.1 PURPOSE

Mr. Shoga's research with regard to WRT-1016 focuses on the identification of synergies and conflicts between system qualities. This information can help stakeholders to identify potential conflicts where overemphasis of a particular quality can have strong negative impacts on other qualities, thus delaying development and increasing TOC. The research can further provide a better understanding of what trade-offs will need to be made and the costs associated with ensuring levels of different qualities for the intended system. A literature study identified current approaches for handling quality interrelationships and the gaps in those approaches [6]. The presentation at the 2021 CSSE ARR provided results of the mapping study and an empirical study showing an approach using the CAST tool to assess quality and correlation analysis as a basis for identifying quality synergies and conflicts.

1.3.2 ON-GOING WORK

Figure 3 depicts the Spearman correlation coefficient matrix for Apache projects. Statistically significant correlations are identified with single asterisked cells for p-values below 0.05 and double asterisked cells for p-values below 0.01. The magnitude of the correlation coefficient indicates the strength of a relationship between the variables. Positive values indicate potential synergies while negative values indicate potential conflicts between the quality metrics. For instance, Robustness-Security and Security-CISQ Reliability are two potential synergies with high correlations. SEI Maintainability - Security and SEI Maintainability - CISQ Performance Efficiency are two potential conflicts highlighted in the figure.

	Tra.	Cha.	Rob.	Eff.	Mai.	Sec.	C.Mai	C.Per	C.Rel	C.Sec
Transferability	1.00	0.37	0.18	0.24	0.36	-0.08	0.36	-0.09	0.02	-0.05
Changeability		1.00	0.86**	0.80**	-0.43	0.73**	0.81**	0.67**	0.71**	-0.27
Robustness			1.00	0.67**	-0.50	0.91**	0.79**	0.63*	0.82**	-0.11
Efficiency				1.00	-0.46	0.66**	0.58*	0.77**	0.59*	-0.08
SEI Maintainability					1.00	-0.63*	-0.27	-0.56*	-0.44	0.19
Security						1.00	0.78**	0.71**	0.91**	-0.03
CISQ Maintainability							1.00	0.48	0.80**	-0.16
CISQ Perf. Efficiency								1.00	0.61*	0.13
CISQ Reliability									1.00	-0.13
CISQ Security										1.00

Figure 3. Synergies and Conflicts Between System Qualities

Mr. Shoga's work involves expanding the empirical study to include additional open-source projects from different ecosystems and domains. This aims at identifying potential effects these may have on the quality interrelationships. The mapping study identified several qualities that tend to be involved in synergies and conflicts which are not covered in the current empirical study; these are planned for further investigation. Finally, the identification of these interrelationships is

based on correlation analysis; additional options are being evaluated for expanding the analysis to identify quality interrelationships.

1.4 SOURCE CODE AS A GRAPH LEARNING TASK-AGNOSTIC REPRESENTATIONS

1.4.1 PURPOSE

Mr. Iordanis Fostiropoulos's research with respect to WRT-1016 is in the use of artificial intelligence Deep Neural Networks (DNN) for predicting TOC and cost estimation. Current deep learning techniques require vast amounts of data for training networks that can be over-parameterized. DNNs have shown to excel in multiple domains.

His work involves introducing improvements to a DNN that will make it possible to train for tasks such as predicting TOC and cost estimation. There is a limited amount of labelled data that can be noisy. Moreover, it is resource intensive to collect additional data. The current work is to improve the efficiency in training DNN with limited size datasets in an unsupervised manner which allows for fine-tuning in the specific task of Cost Estimation post-hoc.

1.4.2 ON-GOING WORK

Currently Mr. Fostiropoulos is working on researching additional performance advantages in Transformers that relate directly with software analysis techniques since they allow for longer sequences of source code to be used for the analysis with higher accuracy since previous methods are limited by the sequence size.

Mr. Fostiropoulos has submitted results to three conferences: Annual Meeting of the Association for Computational Linguistics, Foundations of Software Engineering, and Empirical Methods in Natural Language Processing. His work is under review.

1.5 THE COST OF DEVELOPING SECURE SOFTWARE

1.5.1 PURPOSE

Ms. Elaine Venson's research with respect to WRT-1016 is one the cost of developing secure software. The benefits of applying security practices *early* in the Software Development Life Cycle (SDLC) are frequently discussed in the literature, Figure 4. Such benefits are considered as motivation in many studies, which state that security can be improved, and the total cost of a software system can be reduced with the appropriate allocation of resources in the early stages of the software project, thus reducing TOC.

The total cost of development is reduced because security defects are found and fixed close to their point of introduction. If the same security issues are left to be found during testing and operation, the costs to repair are much higher. This statement is drawn as an analogy with known studies in Software Engineering, which observed the realization of cost savings when problems are detected and fixed early in the lifecycle. Besides considering the savings in security patching, authors often relate benefits with avoiding risks relative to vulnerabilities. Researchers ponder that more secure software implies less losses with down-time and recovery costs from attacks, i.e., operational costs.

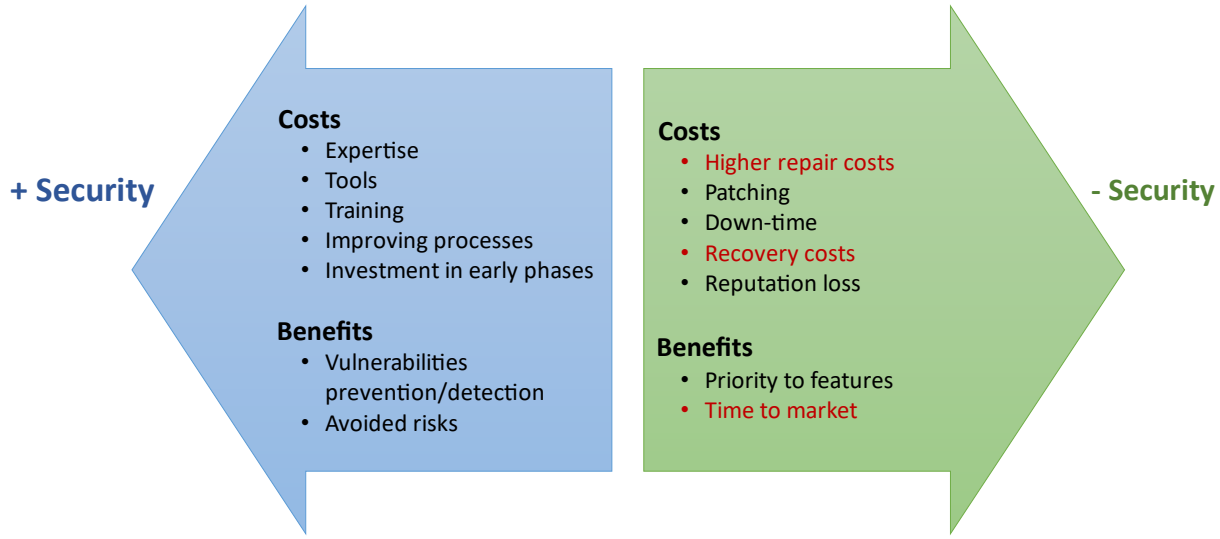

Figure 4. Impact of Security on TOC

Market value is also cited, as software products delivered with vulnerabilities may cause customer dissatisfaction, reputation loss, and sales loss.

An important question to be answered in the context of the cost-effectiveness of developing secure software is how much to invest in order to get the benefits. In secure software development, security competes for a slice of a limited budget - adding more security usually means removing features from the scope. While the growing field of Software Security provides technical solutions to address current security problems, financial issues are still a barrier to their effective introduction in projects. The few proposed cost models for security effort do not consider security practices and were not properly validated, challenging the resulting estimates, which, in turn, hinders cost-effectiveness analysis and resource planning for software projects.

Given the current context of secure software development, this research aims to examine the effects of applying software security practices to the software development effort. The research questions are centered on determining if there is an increase/decrease in software development effort caused by the incorporation of security practices. Understanding the influence of the security practices will allow for improved project estimation and planning, which will, in turn, ensure the provision of proper resources to build secure software, less susceptible to cyber-attacks.

1.5.2 ON-GOING WORK

In a previous phase of this research, a new measurement scale for secure software development was established. Such a scale provided the basis for the data collection and calibration of a proposed software security cost model, see Table 1 and Table 2 below.

Table 1. Security Scale Ratings - One Line Summary

	None/Ad-hoc	Basic	Moderate	Extensive	Rigorous
One Line Security Scale Summary	Security-related activities for requirements, coding, and testing nonexistent.	Basic security-related activities for requirements, coding, and testing. Typical security functional features. Regular use of static analysis tools to detect security defects within the project.	Moderate security-related activities for requirements, design, coding, and testing. Additional security features (audit/log, cryptography). Identification and controlled update of third-part components' security patches. Routine use of static analysis and penetration testing tools. Security V&V activities conducted by an independent group.	Complex security requirements and threat modeling. Advanced secure-by-design security features. Extensive adversarial testing and security design/code review. Security assessment of third-part components and timely security patches updates. Thorough use of statics analysis, black-box, and penetration testing tools. V&V activities conducted by an independent group at the organization level.	Extreme security requirements and threat modeling. Container-based approaches to advanced security features. Exhaustive adversarial testing, security design/code review, deep-dive analysis penetration testing, and use of formal methods throughout the lifecycle. Third-part components rigorously assessed and updated by a security science team. Maximal use of tools for static analysis, penetration testing, and black-box security testing. Use of formal verification and custom developed V&V tools. Security V&V activities conducted by an outside certified company.

Table 2. Security Scale Ratings - Detailed Description

Group	None/Ad-hoc	Basic	Moderate	Extensive	Rigorous
Security Requirements and Design	No security requirements specified.	Basic security requirements and features. Basic threat modeling.	Moderate security requirements and additional security features (audit/log, cryptography). Moderate threat modeling.	Complex security requirements, advanced secure-by-design security features middleware development. Threat modeling with specific attackers' information.	Extreme security requirements, container-based approaches for advanced security features development. Rigorous threat modeling.
Secure Coding and Security Tools	No secure coding and no use of static analysis tool.	Basic vulnerabilities applicable to the software will be prevented with secure coding standards and/or detected through basic use of static analysis tools.	Known and critical vulnerabilities applicable to the software will be prevented with secure coding standards and/or detected through routine use of static analysis tools.	Extensive list of vulnerabilities and weaknesses applicable to the software will be prevented with secure coding standards and/or detected through extensive use of static analysis and black-box tools.	Very extensive list of vulnerabilities and weaknesses applicable to the software will be prevented with secure coding standards and/or detected through rigorous use of static analysis and black-box security testing tools with tailored rules. Employ formal methods in coding.

Group	None/Ad-hoc	Basic	Moderate	Extensive	Rigorous
Security Verification and Validation	None.	Basic adversarial testing and security code review. Basic penetration testing. Security V&V activities conducted within the project.	Moderate adversarial testing and security code review. Routine penetration testing. Security V&V activities conducted by an independent group.	Extensive adversarial testing and security design/code review. Frequent and specialized penetration testing. Security V&V activities conducted by an independent group at the organizational level.	Rigorous adversarial testing and security design/code review. Exhaustive deep-dive analysis penetration testing. Use of formal verification and custom developed V&V tools. Security V&V activities conducted by an outside certified company.

Experts' estimation and project effort data are being collected and analyzed to build a model and quantify the factors that affect software development effort, with a focus on security. An Online Delphi technique was applied to collect expert opinion from security experts that were invited from the Software Security group from LinkedIn. Two rounds of the Delphi were performed, collecting estimates for the productivity range of the scale for the three groups of security-related activities - Requirements & Design, Coding & Tools, and Verification & Validation. Data from actual projects, containing effort, size, security level and other cost driver are currently being collected from companies in Brazil. Based on these sources of data, a model to predict effort for different levels of security is being built and validated.

Ms. Venson's paper on the topic was one of the top papers in the recent Empirical Software Engineering and Measurement conference [7].

CONCLUSION

The research focus of WRT-1016 was reducing TOC and schedule. Maintainability was identified as a system quality that is key in reducing 75% of most system's life cycle costs. One research approach sought to measure the technical debt, Code Smells, carried forward in computer code using SQUAAD and CAST tools. Using the technology developed for the SQUAAD tool, another area of research examined the number of change categories in a single software commit. It was found commits with more than two change categories often resulted in uncompilable code which in turn created rework and more cost. Ten different quality metrics were measured on open-source software to determine synergies and conflicts. Identification of potential quality conflicts, where overemphasis of a particular quality can have strong negative impacts on other qualities, can delay development, create rework, and increase cost. Another research focus was assessing the cost of developing secure software early in the life cycle. The total cost of a software system can be reduced with the appropriate allocation of resources for developing secure software in the early stages of the software project. All of these research directions show benefit in reducing development and maintenance costs thus reducing TOC and schedule.

PROJECT TIMELINE & TRANSITION PLAN

This is the final report for the Research Task (WRT-1016) – "**Reducing Total Ownership Cost (TOC) and Schedule**." It was the first year of a proposed 3-year project: Phases 2 and 3 are option years. Phase 2 plans included:

- Continue collaboration with CAST in integrating CAST and SQUAAD capabilities.
- Continue to explore use of fuzzy analytics in qualities analysis.
- Workshops on cost of security model and its data collection instrument at the virtual 35th COCOMO Forum October 26-27, including capture of the size of security extensions.
- Continued deep dive in exploring CISQ Reliability and Security synergies and conflicts via Common Weakness Evaluations.
- Continue learning CAST and executing more analysis using both CAST and the SQUAAD graphical user interface for executing a DoD SQUAAD private-cloud standalone server.

APPENDIX A: ACRONYMS

AI	Artificial Intelligence
ARR	Annual Research Review
CISQ	Consortium for Information Technology Software Quality
COCOMO	Constructive Cost Model for software cost estimation
CSSE	Center for Systems and Software Engineering
QRS	Quality, Reliability, and Security
SDLC	Software Development Life Cycle
SEI	Software Engineering Institute
SERC	Systems Engineering Research Center
SLOC	Source Lines of Code (computer code)
SQUAAD	Software Qualities Understanding by Analysis of Abundant Data
SSRR	SERC Sponsored Research Review
TOC	Total Cost of Ownership
USC	University of Southern California

APPENDIX A: LIST OF PUBLICATIONS FROM TASK (SEP 2019 – JAN 2021)

[1] Boehm B, Behnamghader P. "Anticipatory development processes for reducing total ownership costs and schedules", The journal of The International Council on Systems Engineering, 2019;1–10.

[2] J. He, S. Min, K. Ogudu, S. Shoga, A. Polak, I. Fostiropoulos, B. Boehm, and P. Behnamghader "The Characteristics and Impact of Uncompilable Code Changes on Software Quality Evolution." IEEE Software Quality, Reliability, and Security (QRS), 2020.

[3] B. Boehm and P. Behnamghader "Large Scale Mission Software Data Exploitation."

Aerospace Corp. Ground Systems Architecture Workshop, 2020.

[4] P. Behnamghader and B. Boehm “Software Quality Understanding by Analysis of Abundant Data (SQUAAD): Towards Better Understanding of Life Cycle Software Qualities.”, SERC Sponsored Research Review, 2019.

[5] P. Behnamghader, B. Boehm et al, “A Scalable and Efficient Approach for Compiling and Analyzing Commit History,” Proceedings, IEEE Empirical Software Engineering and Measurement, 2020.

[6] M. Y. Shoga, C. Chen and B. Boehm, "Recent Trends in Software Quality Interrelationships: A Systematic Mapping Study," 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2020, pp. 264-271.

7] Venson, E., Alfayez, R., Marília M. F., G., Rejane M. C., F., Boehm, B., 2019. The Impact of Software Security Practices on Development Effort: An Initial Survey, in 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Presented at the 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–12.

Other Task-Related Publications

[8] Venson, E., Guo, X., Yan, Z., Boehm, B., 2019. Costing Secure Software Development: A Systematic Mapping Study, in: Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES ’19. ACM, New York, NY, USA, p. 9:1-9:11.

[9] Venson, Elaine. “The Effects of Required Security on Software Development Effort.” Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, Association for Computing Machinery, 2020, pp. 166–69.

[10] Venson, Elaine, et al. “Costing Secure Software Development: A Systematic Mapping Study.” Proceedings of the 14th International Conference on Availability, Reliability and Security, ACM, 2019, p. 9:1-9:11.