Walden University
ScholarWorks

Walden Dissertations and Doctoral Studies

Walden Dissertations and Doctoral Studies Collection

2020

Exploring Software Testing Strategies Used on Software Applications in the Government

Angel Diane Cross
Walden University

Follow this and additional works at: https://scholarworks.waldenu.edu/dissertations

Part of the Databases and Information Systems Commons

This Dissertation is brought to you for free and open access by the Walden Dissertations and Doctoral Studies Collection at ScholarWorks. It has been accepted for inclusion in Walden Dissertations and Doctoral Studies by an authorized administrator of ScholarWorks. For more information, please contact ScholarWorks@waldenu.edu.

Walden University

College of Management and Technology

This is to certify that the doctoral study by

Angel D. Cross

has been found to be complete and satisfactory in all respects,
and that any and all revisions required by
the review committee have been made.

Review Committee
Dr. Steven Case, Committee Chairperson, Information Technology Faculty
Dr. Gail Miles, Committee Member, Information Technology Faculty
Dr. Jodine Burchell, University Reviewer, Information Technology Faculty

Chief Academic Officer and Provost
Sue Subocz, Ph.D.

Walden University
2020

Abstract

Exploring Software Testing Strategies Used on Software Applications in the Government

by

Angel D. Cross

MS, Strayer University, 2008

BS, Hampton University, 2001

Doctoral Study Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Information Technology

Walden University

March 2020

Abstract

Developing a defect-free software application is a challenging task. Despite many years of experience, the intense development of reliable software remains a challenge. For this reason, software defects identified at the end of the testing phase are more expensive than those detected sooner. The purpose of this multiple case study is to explore the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. The target population consisted of software developers from 3 government contracting organizations located along the East Coast region of the United States. Lehman's laws of software evolution was the conceptual framework. The data collection process included semistructured interviews with software developers (*n* = 10), including a review of organizational documents (*n* = 77). Thematic analysis was used to identify patterns and codes from the interviews. Member checking activities were triangulated with organizational documents to produce 4 major themes: (a) communication and collaboration with all stakeholders, (b) development of well-defined requirements, (c) focus on thorough documentation, and (d) focus on automation testing. The results of this study may contribute to information about testing strategies that may help organizations improve or enhance their testing practices. The results of this study may serve as a foundation for positive social change by potentially improving citizens' experience with government software applications as a result of potential improvement in software testing practices.

Exploring Software Testing Strategies Used on Software Applications in the Government

by

Angel D. Cross

MS, Strayer University, 2008

BS, Hampton University, 2001

Doctoral Study Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Information Technology

Walden University

March 2020

Dedication

I dedicate this study to my family. Thanks to my parents, my brother, and most of all, my husband, Eddie. There is absolutely no way in the world that I could have completed this program without your continued support and, most of all, your prayers. Thank you all for cheering me on as my lifelong dream has manifested into a reality. It is never too late to make your dreams come true. I know that I have been unavailable on many occasions while I worked on my research study; in the end, it paid off. You all have been very supportive and understanding, and I am forever grateful. As the first in the family to achieve a doctoral-level degree, I hope to have made everyone proud. To my nephews Justin and Jordan, my niece Robyn, and my cousins Tyrie and Tyler, you are the next generation of leaders. Remember, education is the key to unlocking the world. To achieve your dreams, remember your ABCs. Finally, I dedicate this study to the memory of my grandmother, who would have been incredibly proud of this accomplishment. I miss you a lot. You will always be in my heart.

Acknowledgments

First and foremost, I want to give honor to my Heavenly Father Jesus Christ, for making all of this possible. I encouraged myself daily by meditating on scriptures, Philippians 4:13, Jeremiah 31:17, and Galatians 6:9.

I am blessed and fortunate to have furthered and completed my doctoral-level studies at Walden University. The education that I received from Walden has added value to my life, as well as enhanced my corporate skills. Second, I would like to thank my committee chair, Dr. Steven Case, for his mentorship, support, and guidance during my doctoral journey. The difficulty level would have been extremely high without his guidance and encouragement. Especially during the times when I was tired and wanted to throw in the towel.

Third, I would also like to thank my second committee member, Dr. Gail Miles, for her valuable comments and feedback on making my study complete. Next, I also would like to thank all the staff, administrators, and instructors who encouraged me along the way. I even would like to extend special thanks to all of the participants in my study who so kindly shared information and their time with me. Finally, to my employer, you helped me make this possible in more ways than you could ever imagine.

Table of Contents

List of Tables

List of Figures

Section 1: Foundation of the Study

Developing a defect free software application is a challenging task due to the occurrence of unknown software bugs or unforeseen software defects. The presence of bugs and defects occur regardless of the guidelines followed in the software development lifecycle. Consequently, effective software testing is essential to the development and delivery of reliable software applications. There is no question that the longer a defect remains undetected, the more expensive it is to fix it. For this reason, software testing can save time and money by reducing software development and maintenance costs. As with any activity that requires human involvement, the outcome is dependent on human factors. The purpose of this study was to explore testing strategies that software developers use to ensure the reliability of software applications in the government contracting industry. Section 1 contains the foundation of the study, background of the problem, the problem and purpose statement, the nature of the study, the research question, the interview questions, the conceptual framework, definition of terms, the assumptions, limitations, and delimitations, the significance of the study, contribution to IT practice, the implications for social change and the literature review. Following a review of the academic literature and professional literature on software testing and Lehman’s laws of software evolution, I provide a conclusion and then transition to Section 2 of the study.

Background of the Problem

In this fast-paced age of Information Technology (IT), the release of reliable and defect free software cannot go unnoticed. As more citizens become computer literate, the use of computers, laptops, mobile devices, and other computer-based products have almost dominated our everyday living. All these devices require some form of software involvement (Batool, 2015). At this moment, software controls every aspect of our daily lives, ranging from mobile communication devices and interaction on social media networks to conducting online banking and monitoring our health. Software is a broad term used to define a set of written instructions that a computer follows to perform a specific task. The development of a defect free software application is a challenging task and is of utmost importance. Tomar and Agarwal (2016) developed an application to identify defective software that can benefit software developers by allocating resources for the release of reliable and defect free software products. As there is interest in the development of defect free software, the idea to avoid rework is the goal. In addition to extending time to use, software defects can have more dire consequences. Reports have showed that software defects wrecked a European satellite launch, delayed the opening of a newly constructed Denver airport for a year, destroyed a NASA Mars mission, and killed four marines in a helicopter crash (Oghenovo, 2014). These incidents explain why software testing is so important to the success of any project.

Software testing is vital to the successful execution of a product. Though an essential activity in the software development lifecycle, software testing is primarily conducted to detect any software defects introduced during various phases of the software development lifecycle (Subramanian, Pendharkar, & Pai, 2017). The main problem is that software testing activities consume a great deal of the time allocated toward the overall costs of software development. Therefore, as technology advances, software testing is more critical today as the potential impact for defective software applications continues to rise.

Problem Statement

Software errors that are discovered at the end of the testing phase and software defects that are found by software end-users are much more expensive to fix than defects that are found at the earliest project phases (Petunova & Berzisa, 2017). Based on a study conducted at Cambridge University, the results concluded that software developers spend nearly 50% of their time diagnosing software errors, which leads to an estimated cost of $312 billion per year (Hamill & Goseva-Popstojanova, 2017). The general IT problem is that software defects impact the reliability of software applications. The specific IT problem is that some software developers lack testing strategies to ensure the reliability of software applications in the government contracting industry.

Purpose Statement

The purpose of this qualitative multiple case study was to explore the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. The targeted population consisted of software developers from three government contracting industry organizations located along the East Coast region of the United States. The contributions of this study may help foster a greater understanding on the part of software developers to improve testing strategies to ensure the reliability of software applications in the government contracting industry. Thus, the research findings might contribute to positive social change by possibly improving the everyday life of citizens, as a result of improvement in the reliability of software applications in the government contracting industry.

Nature of the Study

The nature of the study is a description and justification of the selection of the study methodology and design. I selected the qualitative method for this study as it addressed the research purpose to explore and understand the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. A qualitative research method is used by researchers to address the ‘how’ and ‘why’ of a story, in ways that quantitative research cannot (Yates & Leggett, 2016). I selected the qualitative research method for this study because I wanted to explore and understand ‘how’ the testing strategies used by software developers in the government contracting industry ensured the reliability of software applications. A quantitative research method is used by researchers to accept or reject a statistical hypothesis (Haegele & Hodge, 2015). I did not select a quantitative research method for this study because the intended focus of the research question is not to accept or reject a statistical hypothesis. A mixed methods research method is used by researchers to collect both qualitative and quantitative data (Stockman, 2015). I did not select a mixed methods research method for this study because the quantitative method has been eliminated. As I reflect on the probable method, the qualitative method is appropriate for this research because it addresses the intended focus of the research question.

I selected the case study design for this study to explore and understand the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. A case study design is a comprehensive method that incorporates multiple sources of data to provide detailed accounts of complex research phenomena in real life contexts (Morgan, Pullon, Macdonald, McKinlay, & Gray, 2017). I selected the multiple case study design because I wanted to investigate the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. A narrative design studies the lives of individuals and provides stories about their lives (De Loo, Cooper, & Manochin, 2015). I did not select a narrative design for this study as understanding the lives of individuals was not the intended focus of the research question. A phenomenological design describes the lived experiences of individuals or a phenomenon (Aagard, 2017). I did not select a phenomenological design for this study because understanding the lived experiences of individuals is not the intended focus of the research question. An ethnography design studies the shared patterns of behaviors, languages, and actions of other cultural groups (Badri, Wolfe, Farmer, & Amin, 2018). I did not select an ethnographic design for this study because shared patterns, languages, and actions of cultural groups are not the intended focus of the research question. As I reflected on the probable designs, the multiple case study design was appropriate for this research because it addressed the intended focus of the research question.

Research Question

What testing strategies do software developers use to ensure the reliability of software applications in the government contracting industry?

Interview Questions

Background Interview Questions

1. Can you tell me about yourself and your current role?
2. How many years of experience do you have as a software developer?
3. How long have you been performing software testing tasks?
4. What type of project(s) are you currently working on?

Interview Questions

1. What is the primary software development methodology you are using?
2. How is software testing organized in your organization?
3. What testing strategies have you used to ensure the reliability of software applications?
4. How do you assess the effectiveness of the testing strategies used to ensure the reliability of software applications?
5. How satisfied are you with the development and testing environments that you have?
6. What challenges have you faced where you find yourself in a disagreement over a software defect?
7. How has these challenges impacted your testing of software applications?
8. What testing strategies do you find the most effective in detecting software defects?
9. How much time is allocated for testing software applications in your organization?

10. What additional information would you like to share about testing strategies that would ensure the reliability of software applications?

Conceptual Framework

The conceptual framework for this study was driven by Lehman’s laws of software evolution. Meir Manny Lehman developed the laws of software evolution in 1968 as a result of an investigation into programming practices within IBM (Godfrey & German, 2014). The analysis prompted a further study of the IBM S/360 operating system and its successor, IBM S/370 (Godfrey & German, 2014). Lehman’s (1996) work on the laws of software evolution, which he devised and refined with Laszlo Belady and other collaborators over many years, continues to influence the study of ‘how’ and ‘why’ software applications change over time. Lehman (as cited in Godfrey & German, 2014) discovered that software developers were becoming increasingly interested in assessing their productivity, which was measured in terms of daily source lines of code (SLOC) and passing unit tests. Lehman observed that productivity was increasing according to the requirements; however, at the same time, software developers appeared to be losing sight of the overall product (Godfrey & German, 2014). Lehman summarized his observations about the evolution of software into eight laws which include: (a) continuing change law, (b) increasing complexity law, (c) self-regulation law, (d) conservation of organizational stability law, (e) conservation of familiarity law, (f) continuing growth law, (g) declining quality law, and (h) feedback system law.

In this study, I explored the testing strategies used to ensure the reliability of software applications. I used the unique lens of Lehman’s laws of software evolution in my study to understand how attributes of the software evolution phenomenon has an impact on my study and software process improvement. I selected the law of continuing change as it suggests that software will become progressively less satisfying to its users' overtime unless it is adapted to meet new needs. Moreover, the law of continuing change suggests that software developers must be aware that if their software does not respond positively to the pressures of the system, that over time, the system will be less appealing to its users (Godfrey & German, 2014). I also selected the law of increasing complexity and the law of declining quality. The law of increasing complexity indicated that software would become progressively more complex over time unless explicit work is completed to reduce its complexity. The law of declining quality indicated that a software system would be perceived as declining in quality over time unless the design is carefully maintained and adapted to new operational constraints. Both laws imply that the changes required to evolve the system to respond to the pressures tend to make the system more complex and lowers its quality (Godfrey & German, 2014). Finally, I selected the feedback systems law. The feedback systems law suggested that as software ages, it tends to become increasingly complicated as a result of the change. The laws of software evolution were relevant to this study because its core components and constructs align closely with those that I explored in the research. I intended to use Lehman's laws of software evolution as a lens to better understand the study.

Definition of Terms

The following definitions are to assist the reader as these keywords occurred within this study:

Acceptance testing: A software testing strategy that performs tests to validate whether the system meets all the specifications and requirements of the customer and provides assurance that the system is working rather than to find errors (Malik, 2017).

Functional testing: A software testing strategy that discovers disagreements between the specification and the actual implementation of the software application (Julia, Vale, & Passos, 2016).

Load testing: A software testing strategy that refers to the practice of assessing the system behavior under a load. A load is a rate of the incoming requests to the system (Jiang, 2015).

Performance testing: A software testing strategy that determines how fast some aspects of the system perform under a predefined workload. It is calculated by analyzing the production, which comes from the application hosted on the server (Khan & Amjad, 2016).

Regression testing: The pragmatic selection of a test suite from tests developed from other parts of the test process (Parsons, Susnjak, & Lange, 2014).

Software defect: A software bug, error, failure, or flaw found inside the structure of computer source code or system that is the result of some programmatical mistake (Deak, Stålhane, & Sindre, 2016).

Software reliability: The chance of failure free software operation for a specified period (Zhu & Pham, 2018).

Software testing: A phase of the software development lifecycle used to improve the quality of developed software (Jayaram & Krishnan, 2018).

Test case: A set of written conditions that examines all aspects of the structure and logic of a software product or software system to validate the functionality of a specific requirement (Gomez, Cortés-Verdín, & Pardo, 2017).

Unit testing: A software testing strategy which the smallest testable parts of a program individually and independently analyzed for proper operation (Buckley & Buckley, 2017).

Assumptions, Limitations, and Delimitations

Any number of phenomena that affect the internal or external factors can influence the research and its outcomes. It is the process of identifying and analyzing these phenomena that establishes credibility. Three categories of phenomena occur in research; they are assumptions, limitations, and delimitations.

Assumptions

I made assumptions based on the requirements of the study. Assumptions are the beliefs, or the preconceptions of the researcher based on instinct or experience that has not been verified by evidence (Holloway & Galvin, 2016). The achievements of certain assumptions or preconceptions assumed to be true but not verified for this study include the following for assumptions. For this study, the first assumption I made was that software developers representing three government contracting organizations along the East Coast region of the United States would be available and willing to participate in this study. The second assumption is that I assumed that participants would provide open, honest, and unbiased responses to the interview questions during the semistructured interviews. My third assumption is that enough participants would be available within the organization for data saturation to be reached. The final assumption is that I assumed the software developers participating in this study would have software testing experience or knowledge and represent themselves accordingly and fit the established qualifying criteria.

Limitations

I identified the limitations based on the requirements of the study. Limitations are the defects, or the deficiencies encountered that are out of a researcher's control (Horga, Kaur, & Peterson, 2014). For this study, the first limitation of the study was the potential bias due to my views on the research question because I was a software tester in the industry, and I worked in the same geographical region as the study. Bracketing is a technique by which researchers set aside their biases and preconceptions to develop a keen awareness of assumptions and expectations (Sohn, Thomas, Greenberg, & Pollio, 2017). Before conducting interviews, I bracketed my own opinions about the subject and followed approved research protocols to ensure that I did not incorporate any personal bias into the research study. The second limitation of this study was that the accuracy of the results would rely on honest answers from the participants. Yin (2018) emphasized that some participants might respond to interview questions because of what they believe the researcher wants to hear. In part, I mitigated bias by using open-ended questions rather than yes-no questions to allow participants to share their experiences and perceptions about software testing.

Delimitations

I developed the scope for collecting data. Delimitations are factors that limit the scope and state the boundaries of the study, but they are all under the researcher's control (Anthonisz & Perry, 2015). The boundaries that I have placed surrounding this multiple case study would constrain it to a specific population and sampling. The first delimitation was geographically limited to only government contracting industry organizations located along the East Coast region of the United States. Furthermore, Holloway and Galvin (2016) added that delimitations are the boundaries of the research showing what is included or excluded. The second delimitation for this study was that the participants must have at least 2 years of software development experience and software testing experience or knowledge. The third delimitation for this study was that participants identify testing strategies to ensure the reliability of software applications in the government contracting industry rather than to determine how to implement the process. Finally, these testing strategies may vary by government contracting organizations based on when and how they were previously performed.

Significance of the Study

Contribution to Information Technology Practice

This study may be significant to IT practice because it may help foster a greater understanding on the part of software developers to improve testing strategies to ensure the reliability of software applications in the government contracting industry. Software development companies across America lose billions of dollars each year due to the performance of poorly designed software applications (Underwood, 2016). This study contributes information about testing strategies that may help other organizations improve or enhance their testing strategies. Furthermore, this study could be a vision for building a better understanding of testing strategies, which in turn could lead to improved testing strategies, reliable software, profit potential, and the creation of more IT jobs. Finally, this study may fill gaps in the understanding and effective practice of IT by identifying testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications.

Implications for Social Change

This study might contribute to positive social change by possibly improving the everyday life of citizens as a result of improvement in the reliability of software applications in the government contracting industry. Software use has become part of daily living (Ogbodo, 2014). Thus, for this study, the findings may help improve the everyday life of citizens by eventually improving the quality of government applications that enable access to government services. If software developers can design reliable software applications, then more citizens may make use of their services, whether it is for accessing websites that provide advice and helpful information about veterans’ benefits, healthcare, news, and information about the NASA space program, or live events occurring at the United States White House. Therefore, the results of this study could provide a better understanding of the testing strategies, which in turn can lead to more government contractors developing more reliable software that may contribute to improving the everyday life of citizens.

A Review of the Professional and Academic Literature

The purpose of this qualitative multiple case study was to explore the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. The focus of the literature review is the research question: What testing strategies do software developers use to ensure the reliability of software applications in the government contracting industry? I explored how researchers applied Lehman's laws of software evolution as a conceptual framework for their study. Next, I explored the testing strategies software developers used to ensure the reliability of software applications while focusing on the government contracting industry.

The literature review was a significant element of the research study. The literature review was a complex and demanding genre to write (see Badenhorst, 2018). More importantly, the literature review demonstrated an exhaustive review of the rich literature using the chosen theoretical or conceptual framework as a lens to research the phenomenon. According to Steinert and Thomas (2016), the literature review was the foundation upon which all strong scholarly work focuses on a specific topic. Overall, the literature review should be a critical analysis of academic studies and authoritative seminal works.

In this section, my extensive review of the literature established a scholarly foundation for the study, while providing critical analysis to the body of knowledge. According to Cope (2014), the purpose of the professional and academic literature review was to show support of the research topic, identify the literature contributing to research, build a more structured conceptual framework, and help to inform the study results. I organized the professional and academic literature review by subject matter and content.

The fast pace of change and the growing complexity of modern-day software makes it hard to deliver a defect free product (Calcagno et al., 2015). Software developers strive to release new software applications every 90 or 120 days, accelerating the software development lifecycle (Brhel, Meth, Maedche, & Werder, 2015). Most companies today have an increased concern with the quality of their delivered software products and services. For this reason, customers demand better quality software forcing organizations to improve their products and services (Al-Dhaafri & Al-Swidi, 2016). Defective software can vary from minor inconveniences to catastrophic loss of finances or even life. For example, in April 2015, a software anomaly affected more than 300,000 traders on the financial markets at the Bloomberg Terminal in London. As a result, it forced the United Kingdom government to postpone a 3 billion-pound debt sale. In 2016, an F-35 fighter plane fell victim to a software defect, making it a challenge to detect correct targets. Further, Nissan recalled more than 1 million of its vehicles from the market because of a software defect identified in the airbag sensor detectors causing fatal vehicular accidents (Mohan & Shrimali, 2017). In these situations, I assume that Lehman’s laws of software evolution are suitable for understanding the phenomenon of this study. The identification of the appropriate testing strategies could help motivate government contractors to develop more reliable software that may contribute to the improvement of the everyday life of citizens.

Software is evolving and requires constant work to develop and maintain. When compared to software development, software testing is tedious work (Deak et al., 2016). Lehman's laws of software evolution as the conceptual framework established the foundation for this qualitative multiple case study. For this purpose, the literature review was organized into sections composed of discussions on Lehman's laws of software evolution, complementary and contrasting theories, software testing, software methodologies, and software testing strategies.

Literature Review Strategy

The literature review was based on a comprehensive search of online information obtained from the following library databases: EBSCOhost, Proquest Central, ACM Digital Library, Computers and Applied Sciences Complete, Computing Database, IEEE Xplore Digital Library, Science Direct, Google Scholar, Sage Journals, Academic Search Complete, and Military and Government collection. These databases contained a plethora of books, dissertations, conference proceedings, and peer reviewed articles. The use of Ulrich's Periodical Directory allowed me to verify that the sources used in my study were peer reviewed.

A combination of phrases and key terms were used as key search words in the databases for related literature on software testing strategies used in the government contracting industry. Such phrases and key terms included: *software testing, Lehman's laws of software evolution, Linux, software evolution, software maintenance, waterfall model, agile model, black-box testing, white-box testing, regression testing, unit testing, integration testing, and system testing.*

The literature review consisted of peer reviewed articles from journals, reports, articles, dissertations, and seminal books with a focus on research conducted within the past 5 years. I used 194 literature review resources, with 95(%) published between 2014 and 2018. Eighty-four percent of the resources used in the literature review were peer reviewed. A detailed summary of these sources is listed in Table 1 below.

Table 1

Statistics for references in Literature Review

Category	Result
Total number of literature review references	194
Total number of peer reviewed literature review references	178
Total number of peer reviewed references within 5 years	180
Total number of doctoral dissertations	4
Percentage (%) of peer reviewed references	0.84
Percentage (%) of peer reviewed references within 5 years	0.95

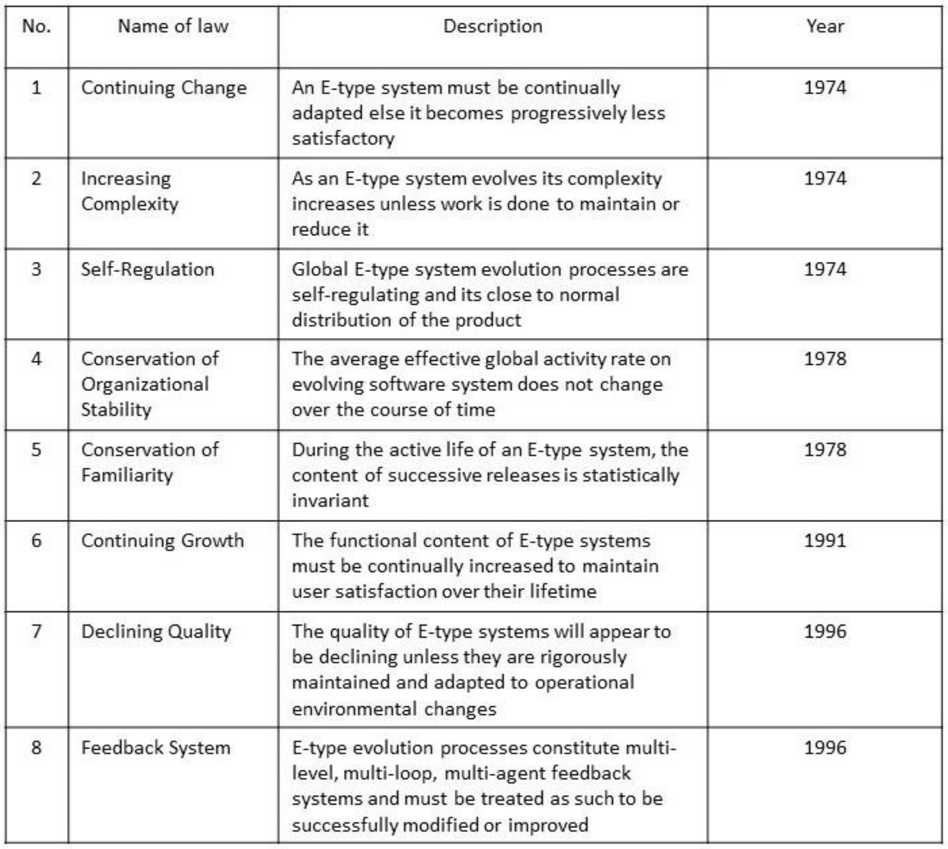
Lehman's Laws of Software Evolution

Lehman's laws of software evolution were selected to drive the conceptual framework for this study. For this reason, the laws defined a balance between efforts driving new developments on the one hand, and efforts that slow down the progress on the other. Meanwhile, Lehman's laws of software evolution encourage a holistic approach when researching any software development problem that would guide this study through the process of exposing the testing strategies used by software developers. By using Lehman's laws as the basis to understand the evolution of software development, the testing of code is an important factor. As it contributes to the overall reliability of the developed code.

Meir M. “Manny” Lehman has influenced how software is created and understood. Reflecting on the mid-1970s, together with Laszlo Belady, Lehman observed and performed empirical work on IBM OS/360 and other large-scale software systems that would later foster a discussion on the understanding of the software development lifecycle by other researchers and practitioners (Godfrey & German, 2014; Oliveria, Santos, Almeida, & Gomes, 2017). The model contributed to improved control of software quality and cost. The empirical studies pioneered by Lehman and Belady gave rise to the eight laws of software evolution (Stol & Fitzgerald, 2015). The eight laws are the result of careful and challenging empirical studies on the evolution of large-scale software systems found in corporate based settings (Aversano, Di Brino, Guardabascio, Salerno, & Tortorella, 2015). Lehman (as cited in Heuser, Fay, Schaefer, & Tichy, 2015) defined the laws of software evolution and identified that systems are subject to the dynamics causing continual changes of software, resulting in increasing complexity. Thus, it is essential to gain a thorough understanding of the way that software evolves.

Software systems are ever changing. When Lehman (1996) examined the software system, he noticed that after the installation of software, the environment changed. To clarify, for software to function correctly, it should adapt to the environment it evolves by sending user feedback (Alenezi & Almustafa, 2015). In 1974, Lehman proposed three laws that were related to software evolution that would shed light further on the analysis (Lehman, 1996). The three laws proposed were the law of continuing change, the law of complexity, and the law of self-regulation (Shehzad & Shaikh, 2017). Belady and Lehman (1976) defined software evolution as the dynamic behavior of programming systems as they are maintained and enhanced over their lifetimes. In other words, as systems in organizations become longer-lived, software evolution is of great importance.

In 1978, Lehman proposed two more laws, the law of invariant work rate and the law of incremented growth (Skoulis, Vassiliadis, & Zarras, 2015). The laws were later renamed to the conservation of organizational stability law and the conservation of familiarity law (Skoulis et al., 2015). Ten years later, the law of continuing growth was proposed and included with the other laws (Skoulis et al., 2015). In 1996, Lehman revised those observations once more to foster the eight laws of software evolution (Skoulis et al., 2015). Lehman demonstrated that it is not easy to evolve software, so, in 1997, the observations relating to the evolution of software systems were published (Kaur & Kaur, 2015). Researchers and practitioners began verifying the validity of the laws through the scope of open-source and industrial software (Oliveria et al., 2017). Once Lehman’s work was published, the laws became known as Lehman’s laws of software evolution and are noted as the law of continuing change, the law of increasing complexity, the law of self-regulation, the law of conservation of organizational stability, the law of conservation of familiarity, the law of continuing growth, the law of declining quality, and the law of the feedback system (Herraiz, Rodriguez, Robles, & Gonzalez-Barahona, 2013). Lehman's laws of software evolution are not just descriptions of the evolutionary process, but they identify the principles in software development. Now, it is crucial to note that Lehman's laws are nothing similar to the laws of physics or chemistry; yet, they were derived from the habits and practices of people and organizations (Bruyn, Mannaert, Verelst, & Huysmans, 2018). Lehman's laws were proposed to form an environment within which the effectiveness of programming methodologies and management strategies and techniques could be evaluated. Figure 1 is an illustration of Lehman’s laws of software evolution (Lehman, 1996).



No.	Name of law	Description	Year
1	Continuing Change	An E-type system must be continually adapted else it becomes progressively less satisfactory	1974
2	Increasing Complexity	As an E-type system evolves its complexity increases unless work is done to maintain or reduce it	1974
3	Self-Regulation	Global E-type system evolution processes are self-regulating and its close to normal distribution of the product	1974
4	Conservation of Organizational Stability	The average effective global activity rate on evolving software system does not change over the course of time	1978
5	Conservation of Familiarity	During the active life of an E-type system, the content of successive releases is statistically invariant	1978
6	Continuing Growth	The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime	1991
7	Declining Quality	The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environmental changes	1996
8	Feedback System	E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to be successfully modified or improved	1996

Figure 1. Lehman's laws of software evolution. From "Metrics and laws of software evolution-the nineties view" by M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, & W. M. Turski, 1997, *Proceedings Fourth International Software Metrics Symposium,* p. 21. Copyright 1997 by IEEE. Reprinted with permission (Appendix D).

Continuing Change (Law 1). The continuing change law is the first law of software evolution. According to Lehman (1996), the continuing change law emphasized the point that a software system must continually adjust to its environment to meet user needs. Generally, it is challenging to differentiate between the general growth law (law six) and between changes in the environment. Since the environment of the real-world is ever changing, systems and software must evolve as the world changes or face becoming less applicable and useful (Lehman, Ramil, Wernick, Perry, & Turski, 1997). Lehman et al. (1997) believed that the continuing change law supports but does not contradict the observations proposed two decades earlier that software evolution is still relevant.

Nevertheless, incremental growth may increase the number of defects. Lehman et al. (1997) believed that evolution continues until it is found to be cost effective to reinstate the system with a recreated version. Therefore, the existence of rippled cycles provided evidence that the growth rate sometimes declines with time. Hence, the observation of this law suggested that large systems are incomplete; yet, they continue to evolve to remain useful.

Increasing Complexity (Law 2). The increasing complexity law is Lehman's second law of software evolution. According to Lehman (1996), as a program evolves, its complexity increases unless there is work to reduce the complexity. Duran, Burns, and Snell (2013) reminded us that the increasing complexity law is difficult to prove since both trends are possible. As an example, agile projects are refactored frequently during the development process (Duran et al., 2013). Refactoring decreases the amount of code complexity through simplification and robust design. Most researchers like Oliveira and Almeida (2016) expected the complexity to increase at a slower rate under agile development because there are regular efforts to reduce it. A non-agile project, on the other hand, the complexity would gradually increase as proposed by Lehman with the possibility for sudden dips, which should correspond to a major refactoring or similar effort (Duran et al., 2013). Some researchers have explored the increasing complexity law and noted that data growth rates decline over time (Duran et al., 2013; Oliveira & Almeida, 2016). Nevertheless, an alternative approach is to measure the source code complexity directly, given the availability of the full source code for each version. So, as evolving software continually changes, the complexity increases unless effort is applied to maintain or reduce it.

Self-Regulation (Law 3). The self-regulation law is Lehman's third law of software evolution. Lehman (1996) found that incremental effort spent on each release remains constant throughout a systems lifetime. Lehman (as cited in Amanatidis & Chatzigeorgeou, 2016) noted that the system evolution process is self-regulating. Furthermore, Lehman noted that over time, any measurements of the system or its process would follow a clear trend, with ripples in either direction that follows a normal distribution (Godfrey & German, 2014). Kaur & Kaur (2015) suggested a balance between what is necessary for change and what is achievable. Thus, to have a constant evolution process, the limitation on the growth rate should be accepted, and productivity should be predictable.

Conservation of Organizational Stability Law 4). The conservation of organizational stability is Lehman's fourth law of software evolution. Lehman (1996) found that the average work rate on an evolving system is statistically invariant. When researchers such as Herraiz et al. (2013) studied the context of the conservation of organizational stability law, they explored the maintenance effort spent on the system. Consequently, it is hard to change the staff who has been working on evolving software. The average global effective rate in evolving software tends to remain constant over a product's lifetime (Kaur & Kaur, 2015). More importantly, Kaur and Kaur (2015) found that the process of evolving software is necessary so that it can be used for an extended period. Therefore, the conservation of organizational stability concludes that adding more resources or effort does not profit the system in a meaningful way.

Conservation of Familiarity (Law 5). The conservation of familiarity is Lehman's fifth law of software evolution. Lehman (1996) found that during the evolution of software systems, the content of successive releases remained consistent because software developers needed to have a thorough understanding of the source code and behavior. Skoulis et al. (2015) reported that users should be familiar with the changes made. Since it remains inevitable that without a familiarity with ‘how’ and ‘why’ the system exists, the effort becomes challenging to implement changes and the ability to understand them. In any event, as systems evolve, all available resources, whether software developers, software testers, or end-users, must maintain proficiency in the content and behavior to achieve satisfactory evolution.

Continuing Growth (Law 6). The continuing growth law is Lehman's sixth law of software evolution. Lehman (1996) found that evolutionary type systems must be continually enhanced to maintain user satisfaction over the lifetime of the system. Godfrey and German (2014) believed that the continuing growth law is relevant to Lehman's second law of evolution, the increasing complexity law. Since growth infers adding new features to software and change often infers adding new code, the size of the software increases to meet user requirements. According to Kaur and Kaur (2015), Lehman’s continuous change and growth laws are essential for maintaining the software for an extended period. These laws further explored that as time progresses, it becomes more complicated and more problematic to add new features as a result of changes and growth. When Kaur and Kaur (2015) analyzed the continuing growth law, they found that the systems are thoroughly tested and functional after each increment. In the end, the lifetime of each increment executes until the overall system decommissions.

Declining Quality (Law 7). The law of declining quality is Lehman's seventh law of software evolution. Lehman (1996) found that poorly designed software leads to the introduction of software defects and incomplete requirement specifications. Godfrey and German (2014) reported that during software evolution, the quality of the software should be appropriately maintained; otherwise, it decays. Per Kaur and Kaur (2015), the software is not liable to wear and tear, but it could prove to be unusable if it is not responsive to always changing user’s needs. Since evolution is necessary for the lifetime of the software, software systems would decline unless maintained and adapted to operational environment changes.

Feedback Systems (Law 8). The final law of Lehman's software evolution is the feedback systems law. Lehman (1996) found that as a software system ages, it tends to become increasingly complicated to change due to the complexity of both the artifacts as well as the processes involved as a result of the change. Although the feedback systems law was the last to be proposed, researchers such as Shehzad and Shaikh (2017) argued that it should have been the first law to have been published. The authors reminded us that in software evolution, feedback comes first, and the themes pervade all others allowing software developers the opportunity to be keenly aware of their software systems that do not respond positively. Thus, to remain relevant and useful to the environment, software developers have to respond to the feedback given by the user community.

This section introduced Lehman's laws and showed that at the heart of software evolution is change. Organizations make enormous investments in their software systems which, is critical to the business. As a result, the majority of the software budget is devoted to changing and evolving existing software rather than developing new software applications. Lehman's laws have been criticized for lacking a solid empirical foundation; however, they help to explain that change is inevitable and not a result of bad programming (Godfrey & German, 2014). As argued elsewhere, the reasoning for exploring Lehman's laws is to show the presence, not the absence of potential software defects. Thus, to accomplish this, software developers must implement testing strategies designed to mitigate the risk of software defects. As Julia et al. (2016) pointed out, software testing ensures that expected business systems and software features behave as expected. As business systems change, the environment changes and software testing is required.

Lehman’s Laws and Software Evolution

As evolution continues, the complexity of an evolving system is inclined to increase unless work is undertaken to control or reduce it. Lehman and Ramil (2002) conducted an exploratory analysis concluding that software evolution involves programming paradigms, approaches, languages, and usage domain. Their exploratory research found that evolution should restrict the programming process artifacts such as specifications, designs, and documentation. In 2003, Grubb and Takang advocated the significance of Lehman’s law of software evolution and proposed similar prerequisites, underlying essentials before attempting software evolution and maintenance analysis.

Meanwhile, Javed and Alenezi (2016) found that software needs to evolve to survive, thus undergoing several changes, including modifications to the other attributes such as software maintainability. In their study, Javed and Alenezi (2016) proposed a technique that would reveal software defects in quality software that is stable and maintainable. According to Tomar and Agarwal (2016), a software defect is an error or deficiency in a software process that occurs due to incorrect programming logic, miscommunication of requirements, the lack of programming experience, and the lack of software testing skills. When Tomar and Agarwal (2016) reported that defective software could lead to a poor-quality software product, they found a software defect prediction problem using Lehman’s laws of software evolution to maintain quality software. Their study found that class imbalance often occurs in software development and other real-world applications, which deteriorates the performance of machine learning. In the end, their study confirmed the law of declining quality.

The laws of software evolution described the trends that occurred in time series. Lehman empirically derived the laws of software evolution, explaining the dynamics and patterns behind the evolution of software in time (Ruohonen, Hyrynsalmi, & Leppanen, 2015). Ruohonen et al. (2015) found that the most significant law suggested that evolving software grows and requires constant testing, maintenance, and development work. According to Ruohonen et al. (2015), the motivation for evolving software originated from the surrounding environment to which various feedback mechanisms remain attached throughout evolution. So, as requirements change, evolving software that does not meet the challenges brought forth by the environment decays. Lehman’s research spanned over three decades within the software process domain that formulated some software evolution laws, later supported by Godfrey and German (2014) and several other practitioners. A clear difference of opinion occurred when Kaur and Vig (2016) sharply criticized Lehman's laws of software evolution as a basis for confirming the law of continuing change, the law of self-regulation, and the law of continuing growth. Their research showed that statistical tests tend to contradict the laws even when passed.

The use of web applications has become essential in our daily lives. Many end-users have used web applications to obtain information to conduct financial transactions to having fun and communicating on social media platforms. When Amanatidis and Chatzigeorgeou (2016) questioned whether Lehman’s laws of software evolution confirmed practice for web applications, the results of the study concluded that the law of continuing change, the law of increasing complexity, the law of self-regulation, the law of conservation of organizational stability, and the law of continuing growth are confirmed. Consequently, the law of increasing complexity and the law of the feedback system did not hold in practice, but the law of declining quality failed. In 2016, Bian, Parande, Koru, and Zhao’s published work criticized Lehman’s laws of software evolution, arguing that smaller modules deserve higher attention levels of testing while focusing on source code inspections. Therefore, the need for software and environment changes originated from the quest to maintain quality software and improving maintainability while ensuring customer satisfaction.

Software merging is essential to the maintenance and evolution of large-scale software systems. For more than two decades, software evolution has endured challenges leading to the issue of software merging, as noted by Bouras and Maouche (2017). In previous work, Bouras and Maouche (2017) explained that the objective of software merging is to compare and merge different versions of software in a reliable way, such as to obtain a new version. For this purpose, Bouras and Maouche (2017) investigated an approach to understanding software evolution by using the compare and merge technique. The results of their research found that once new requirements for an existing system are added incorrectly, the system then merges into a new operating system. Goltz et al. (2015) supported the compare and merge technique suggesting that it represented a challenging undertaking. The goal was to develop an innovative method for the continuous evolution of software at a level that is cost effective than merging code. Bouras and Maouche (2017) did not support Goltz et al. (2015) statement, arguably noting that comparing and merging one thousand lines of code is not an efficient way to evolve software since software requirements change rapidly and become obsolete. Charrada, Koziolek, and Glinz (2015) claimed that updating software requirements is a manual task that is expensive and time consuming. Nevertheless, there is still a need for quality software testing. On the whole, software evolution should not compromise software quality.

Software evolution is inseparable from software maintainability, an essential software quality attribute that deteriorates with changes that continue to get integrated throughout the evolution cycle (Braga de Vasconcelos, Kimble, Carreteiro, & Rocha, 2017). For forty years, Lehman and his colleagues hypothesized and tested a series of software evolution laws explaining the universal aspects of software system behavior (Skoulis et al., 2015). Altogether, Lehman's laws of software evolution explained the forces that drive new software development on the one hand and the forces that slow down the development progress on the other hand. Since Lehman's laws are assumed to observe all the changes during the software evolution process, some empirical observations of studying the development of the open-source system seem to challenge most of Lehman's laws of software evolution (Stol & Fitzgerald, 2015). In their work, Braga de Vasconcelos et al. (2017) discussed how software evolution correlated with the software lifecycle, suggesting that software evolution was an essential, feedback-driven, property of software.

Software evolution is essential to real-world software applications. As requirements change, the needs of software change; otherwise, it becomes less useful. So, for software to be used for an extended period, it must evolve. The general idea of open source software is to allow public access to source code so that any user can use it, modify it, or redistribute it in the revised form. However, some empirical observations of studying the development of open-source systems appear to challenge some of Lehman's laws (Alenezi & Almustafa, 2015; Guan, Peng, Perneel, & Timmerman, 2016; Stol & Fitzgerald, 2015). After all, there is no broad support that exists for all the laws across the various empirical studies of open-source systems. Through the years, most researchers have analyzed open-source software evolution from several angles, including growth, quality, and group dynamics (Saini, Mehmi, & Chahal, 2016). When Kaur, Ratti, and Kaur (2014) studied two open-source software cases developed in C++, their study found that the continuing change law, the increasing complexity law, and the continuing growth law can be determined using different metrics. In contrast, the self-regulation law, the conservation of organizational stability law, and the conservation of familiarity law are difficult to evaluate on open-source software. Nevertheless, the declining quality law and the feedback systems law required further observation of open-source software.

Software systems must evolve to satisfy new demands. Lehman (1996) inferred that software systems must evolve; otherwise, there might be a risk of losing market share to competitors. According to Wohlin, Smite, and Moe (2015), Lehman categorized all software systems to fit into one of three types: E, S, or P. Lehman (as cited in Skoulis et al., 2015) used the keyword E-type to describe real-world systems as they evolve. Lehman (1996) observed that E-type software is part of the changing environment, especially when the system changes in response to a new user requirement or change request. When Lehman (as cited in Skoulis et al., 2015) used the keyword S-type (or static type), systems are comprised of static, formal, and correct requirements that have easy to understand solutions. Moreover, Lehman (as cited in Skoulis et al., 2015) used the keyword P-type (or practical type) to define precise requirements; in other words, the solutions are not challenging to comprehend. Hence, Lehman's laws suggested that over time and due to changes and growth, software systems become complex and complicated to add new functional features. Thus, software evolution, which is the continual process of change, is required to maintain updated software with the changing operational domain (Klein, Polin, & Sutton, 2015). Therefore, frequent software change is needed for stakeholders to remain at an acceptable level in a changing world (Klein et al., 2015).

The ability to manage change is critical. Kour and Singh (2016) believed that software evolution is a continuous process that includes activities like software improvement, adaptation, and correction that arise after the operational release of the software. In 2016, Kour and Singh proposed a theory to understand software evolution based on the quantifiable concept of evolvability. The theory included the study of software product quality, the software evolution process, and their relationships with the organizational environment. More specifically, Kour and Singh (2016) assessed the opportunities for analyzing and measuring evolvability at predesign, architectural, and source code phases in the software development lifecycle. In work carried out by researchers Kaur and Vig (2017), Lehman's laws of software evolution have been researched and validated; but there exist few studies that verified the laws for databases in open source. Most of all, the research carried out by Kaur and Vig (2017) explored the properties of growth for database evolution by analyzing Lehman's fifth and sixth law of software evolution, conservation of familiarity law, and the continuous growth law on three open-source databases. Their research found that Lehman's laws of continuous growth and conservation of familiarity applied to open source java databases such that the laws validated all the datasets involved.

Software libraries evolve to adapt to a changing environment. The evolution process requires responding to customer needs, resolving software defects, or addressing other maintainability concerns (Amanatidis & Chatzigeorgeou, 2016). Software is primarily designed, modified, and maintained by humans. Lehman explored evolution over time and noticed that software becomes complex and hard to add new features (Godfrey & German, 2014). In his published observations, Lehman discussed the first and second laws as they relate to software evolution. Lehman (1996) indicated that software systems are written to reflect real-world activities, since the laws need to be adapted, or they will become useless. In work carried out by Kebir, Borne, and Meslati (2017), they supported Lehman's laws of software evolution, noting that the laws are still valid. Though this may be true, increased software complexity is linked to poor software maintainability, leaving a negative impact on the design quality and future changes. The complexity of software is one of the main problems where software defects go undetected (Kebir et al., 2017). Thus, viewing software as a complex evolving system may encourage software developers to accept software diversity as a vital source of robustness.

Software maintenance may deteriorate the quality of the software. One of the main ways to reduce the undesired effects of maintenance is by code refactoring or restructuring existing code. Hora, Silva, Valente, and Robbes (2018) reported that practitioners implement code refactoring; after all, it could be unnecessary work. Furthermore, Hora et al. (2018) found that between 10-21% of changes at the method level in 15 large Java systems were untracked. Dos Santos-Neto et al. (2015) reminded us that code refactoring is a technique used to improve the quality of software without changing the design and behavior of existing software. When Kula, Ouni, German, and Inoue (2018) introduced the technique of code refactoring, they advocated that the code base may increase the complexity and maintainability efforts of Lehman's second law of evolution. In contrast, Singh and Kaur (2017) warned of the code smell activity that identified a more in-depth problem residing in the source code. While not all code smell activities are relevant to the goals of the system or its health, a thorough analysis can reveal the software defects through everyday software development activities, such as program comprehension, maintenance, and evolution.

The complexity of software is a crucial aspect of software evolution research. Wahler, Drofenik, and Snipes (2016) reported that software developers design a great deal of complex software without any formal training or knowledge of Lehman's laws of software evolution. For this reason, multiple modules are inclined to software defects because software developers have difficulty understanding the laws. Heuser et al. (2015) reported that evolving the design of an industrial system is already difficult and complex. However, researchers like Ramos, Kreutz, and Verissimo (2015) emphasized that the more complex the software development effort is, the more challenges software developers face to maintain it. Henceforth, poor design choices may result in complex software that is expensive to support and maintain. In any event, the lesser the complexity level, the easier it is to measure the various other factors of the code.

Quality software is essential to customers, and most importantly, to businesses of all kinds. The need for changing software would always occur because of new customer requirements, company changes, and even technological advances. Shehzad and Shaikh (2017) suggested that software evolution is the sequence of changes to a software system over its lifetime. Software evolution is very complicated because of the constant changes in the environment. Godfrey and German (2014) compared the work of Lehman's laws of software evolution to other types of evolution, offering insight into questions of both the science and engineering fields, as they further examined the forces that would shape change.

In contrast, Maisikeli (2016) found that after the development of a software system, the likelihood that some evolution may endure because of business changes, defect-correction exercises, or preventative maintenances would satisfy the overall performance of the system. Maisikeli (2016) found that even a minor change in an object-oriented software system might produce significant software defects causing rippling-wave effects across an entire software system. Before software is released, it could go through significant changes. Hence, the most crucial goal is to validate code designs through software testing (Parampreet & Rajeev, 2018). Once software evolves, a great deal is learned to provide an opportunity to collect data for future analysis.

The tremendous growth in IT companies was a result of bringing quality software products to the market. According to Haitzer, Navarro, and Zdun (2017), software changes would still exist because of customer needs, market changes, and technology advances. While there may be justification, the choice to develop reliable software would always exist. Almugrin, Albattah, and Melton (2016) argued that requirements change with time, and software systems must frequently be updated to support the changes. The need to release software products on time and the need to satisfy customer satisfaction often compel software companies to release software at the optimal time. Vora (2015) noted that no software does not have a software defect. Even if there are no software defects, it does not prove that they do not exist. Therefore, designing quality software from the start with a suitable process improves product quality.

The role quality plays in the development of software is crucial to the success of any company. So, to deliver quality software, the primary goal is to create the exact requirements and extract those that cause failure (Purohit & Sharma, 2016). Researchers Purohit and Sharma (2016) suggested the use of Quality Function Deployment (QFD) as one of the methods that offer specific software requirements. Purohit and Sharma (2016) used this method to analyze the tool for the evolved software system by comparing features of the programming languages. The focus was to explore the comparison of the programming languages by using decisive factors and functions to analyze an evolving system. The maintenance of the software architecture after deployment is quite complex because of frequent changes in the environment and requirements. Before the changes are modified, the software architecture must evolve. Huckabee (2015) argued that knowing what to build before development commences reduced rework. The goal is to support the decisions software developers make after the deployed software. The deployment process focuses primarily on the maintenance phase because the software requires frequent changes.

The scope of software visualization is to assist users with analyzing software through the lens of visual resources. According to Novais, Santos, and Mendonca

(2017), software visualization can be most effective when used to understand a significant amount of data produced during software evolution. Hence, software developers need to understand the vast amounts of data required to maintain quality software. Novais et al. (2017) used their research as a basis to present an experimental approach that exploits the advantages of combining multiple visual strategies of software evolution. The themes of the study confirmed that combined visualization strategies perform better regarding correctness and analysis time. The goal of software visualization is to help users to understand the software through the use of visual resources. Thus, software visualization could be used to analyze and understand the large amounts of data produced during software evolution (Alnabhan, Hammouri, Hammod, Atoum, & Al-thnebat, 2018). According to Alnabhan et al. (2018), visualization improved the understandability of the software system efficiently and effectively. It, therefore, enhanced different stages of the software lifecycle, including maintenance, reuse, re-engineering, and evolution. Wang, Zheng, Zhang, Zhou, and Dong (2018) used software visualization in their exploratory to identify the evolution of research. They found that the visualization maps for the evolution of research hotspots increased with time.

Software systems must continually evolve to implement new requirements or new environments. Most importantly, building quality software is an essential goal for software developers (Okwu & Onyeje, 2014). When Okwu and Onyeje (2014) explored Lehman's laws of software evolution, their research showed that a software system must be frequently modified. Lehman’s laws outlined the principles that are common to all, whether small, large, or E-type software systems. Stol and Fitzgerald (2015) reported that researchers should conduct studies that produce evidence for practitioners to make well informed decisions regarding Lehman's laws of software evolution. Decan, Mens, and Grosjean (2018) proposed novel metrics to capture the growth, changeability, reusability, and dependency of networks to analyze and compare software evolution. Thus, the findings would assess the quality of the networks.

Software evolution is an essential aspect of organizations. Today's software organizations must invest large amounts of money in maintaining software for the systems used (Munir, Runeson, & Wnuk, 2018). Those systems should be reliable, testable, and maintainable to support the software that it executes. Haitzer et al. (2017) suggested that it is typical for software developers to only pay attention to code designed for open-source projects, where the focus on planning and modeling is often non-existent. According to Bahamdain (2015), open-source software is publicly accessible and provides a lot of services and products to various companies, educational and government organizations, including the White House. The issue that most software developers have with open source software is that access to the source code is permissible, allowing anyone to read, analyze, and modify for improvements (Javed & Alenezi, 2016). In the scope of things, software defects may linger if the software does not evolve (Bergmane, Grabis, & Zeiris, 2017). So, this means that experienced software developers are responsible for maintaining the functionality of the system by designing quality software defect free code.

This section on Lehman’s laws and software evolution has shown that software evolution is still evident as business needs change and the criteria for satisfaction changes. Lehman challenged the commonly held view that evolution is essential to real-world software (Godfrey & German, 2014). Since the software is the basis upon which many businesses operate, it is paramount to the success of any business to have newly developed or recently modified software tested for defects. To accomplish this, software developers must fix a discovered software defect sooner than later, understanding Lehman’s laws of software evolution. When software defects are found late in the development process, it would likely take more time for the software developer to unravel the code from around the defect. As Ivanov, Reznik, and Succi (2018) pointed out, the goal of software testing is to ensure that software evolution does not break the existing functionality. Therefore, the software developer would need to validate if a new defect or a new software release does not violate the existing code.

Lehman’s Laws of Software Evolution and the Linux Kernel

In this section, I provided some background regarding the Linux kernel and how it relates to Lehman's laws of software evolution. The Linux kernel, which is an essential part of the Linux operating system, was initially written by a University of Helsinki computer science student, Linus Torvalds, and later released and maintained under the Free Software Foundation's GNU General Public License (GNU GPL), which manages free software (Bansal, Kellis, Kordi, & Kundu, 2018). According to Karpowicz, Arabas, and Niewiadomska-Szynkiewicz (2018), Linux is efficient and reliable and is available in many versions, providing support for most modern-day computer hardware. Though people can freely download and distribute Linux, some software developers charge fees, which they justify by providing customer support for their versions of the operating system (Rigoni, Manduchi, Luchetta, Taliercio, & Shroder, 2018).

Software systems are complex entities. They must evolve statistically and dynamically regarding size and complexity. Olatunji, Oladele, and Bajeh (2017) found that Lehman's continuing change law and increasing complexity law apply in the context of system builds. For more than three decades, thousands of software developers have contributed more than 18 million lines of code to the Linux kernel (Bagherzadeh et al., 2018). According to Bagherzadeh et al. (2018), the Linux kernel forms the central part of various operating systems that are used by a multitude of end-users. The Linux kernel is a sophisticated system and can be used by software developers who want to improve the design of their work. Further, the Linux kernel provided its services to an application through system calls. Hatton, Spinellis, and van Genuchten (2017) examined the growth rate of the Linux kernel. Their research showed that over time, the growth rate is linear or decrease according to Lehman's laws of software evolution. As noted by Olatunji et al. (2017), Lehman studied seven commercial large software systems and came out with the laws of software evolution. Still, there remain significant differences between industrial systems and open-source software. For example, the development of open-source software is performed by software developers around the world. Thus, the development of open-source software and its evolution or maintenance activities are done at the same time, which happens at different times in commercial systems.

This section on Lehman's laws of software evolution and the Linux kernel has shown that Linux is the most used open-source operating system designed to run efficiently on most modern-day computer hardware. Linux allows end-users to control a platform's direction. As Olatunji et al. (2017) pointed out, Lehman's continuing change law and complexity law supports the Linux operating system in the context of system builds. Empirical studies have demonstrated that a strong incentive for developing open-source software is to solve a technical problem (Bagherzadeh et al., 2018; Bansal et al., 2018). When open-source software is given away for free to software developers who write code, there is generally no rigorous software testing approach taken. Therefore, software testing is narrowed down to whether the results look about right. Moreover, when testing a software system, it requires having an understanding of what and how to test. To accomplish this, using best practices such as integration testing and unit testing, along with the proper tools, might implement testing strategies designed to mitigate software defects. Since software testing is essential, it is often neglected due to its complexity and time-consuming process (Rigoni et al., 2018).

Lehman's Laws of Software Evolution and Software Maintenance

In this section, I provided some background regarding software maintenance activities and how they relate to Lehman's laws of software evolution. The success of software requires constant change and maintenance. Software evolution is a field that examines the application of software maintenance activities, changes in software processes, and the resulting evolved versions of the software (Granli, Burchell, Hammouda, & Knauss, 2015).

A large part of software maintenance is software comprehension, which uses a massive amount of time and effort (Khatiwada, Tushev, & Mahmoud, 2018). According to Khatiwada et al. (2018), as much as 70% of the total lifetime cost for software is for maintenance related activities. The need for software change has increased due to the rapid growth of software development. Cashman and Rosenblatt (2014) reported that maintenance expenses vary significantly during the system's operational life. Further, they explained that the maintenance activities include changing programs, procedures, or documentation to ensure that the correct system performance, and causing the system to operate more efficiently.

When performing maintenance of a software system, it is not possible to make changes without having a complete understanding of the system and the interactions within that system. According to Granli et al. (2015), the four categories of maintenance include: corrective, adaptive, preventive, and perfective. When Granli et al. (2015) talked about corrective maintenance, they reported that it diagnoses and corrects errors for functionality. Moreover, adaptive maintenance makes the system more comfortable to use; whereas, preventive maintenance requires an analysis of areas where a problem is likely to occur. Although perfective maintenance can improve system reliability, it involves changing an operating system to make it more efficient (Granli et al., 2015). Practitioners involved in software evolution consciously or unconsciously confront some of the constraints imposed by the laws of software evolution that Lehman introduced during the 1970s (Camilo, L'erario, Pagotto, & Fabri, 2018). Further, Coelho, Valente, Silva, and Shihab (2018) carried out research to alert end-users about the risks of using projects that were unmaintained or sparsely maintained. They argued that Lehman's laws of software evolution deal with stable or controlled environments. In the end, their research showed that 75% of the studied projects are unmaintained.

This section of the study explored Lehman's laws of software evolution and software maintenance. Software maintenance and software evolution are a continuous process in the software development lifecycle to repair existing faults, enhance platform compatibility, and increase user satisfaction. While successful software requires constant change triggered by evolving requirements, software evolution is inevitable (Khatiwada et al., 2018). As Coelho et al. (2018) pointed out, a large portion of the software maintenance budget is devoted to time and effort. Thus, it is necessary to have a good understanding of the software system as a whole to make the required changes effective. To accomplish this, software developers should employ systematic, continuous performance regression testing strategies to reveal software defects in the early stages of the testing process. Thus, due to high overhead charges, performance regression testing is too expensive. To this end, software applications are not thoroughly tested to ensure the reliability of the product.

Complementary and Contrasting Theories

Software testing is understood to be a bug hunting activity. According to Alhammad and Moreno (2018), software testing is the main activity for evaluating and executing software to discover defects. Software testing is an entity of software engineering. For this reason, software testing is performed to ensure that the developed software application is working as it is defined and intended to operate (Dalal & Hooda, 2018). Software testing is the most common approach in the industry to validate the correctness of the software. Due to the nature of the study, it follows that Lehman's laws of software evolution were selected as the conceptual framework to help understand the phenomenon and to explore the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. Hence, the following section breaks down theories that either support or contrast the selected theory for the study.

As there are many theories that I could choose from, as the researcher, I examined each theory and determined the relevancy for answering the central research question. According to Khachaturian et al. (2018), theoretical articles are written for a diverse audience, so that the central research question and any linkages with existing ideas to the conceptual frameworks or theories are easily understood. This research falls under the broad theoretical area of software development, testing, and maintenance. As for this research study, I reviewed complementary and contrasting theories as conceptual frameworks and how they are applied to case study research.

Software Testing Theory. Software testing is a significant part of the software development lifecycle, although it is the primary method for detecting software defects. One supporting theory is software testing theory. Lemos, Silveira, Ferrari, and Garcia (2018) noted, one of the characteristics of the software testing theory is that it is more reliable than the theories previously introduced. Lemos et al. suggested that the software testing theory might improve programming skills as a result of generating quality software. Clarke, Davis, King, Pava, and Jones (2014) argued that more exposure to software testing practices, tools, and better training for software developers might contribute to quality software. Further, Beppe et al. (2018) emphasized that software testing is used to evaluate and improve software quality. For this reason, software testing confirms if the software does what it is intended to do, which is to identify problems in the software before it is released. While there is no one to guarantee the best practices of the software testing theory, many researchers are still conducting continual work as much is built on wishful thinking (Beppe et al., 2018; Lemos et al., 2018; Yao & Liu, 2018).

Gerhart-Howden-Duran Testing Theory. Another theory that was considered for this study, but was not chosen, was the Gerhart-Howden-Duran testing theory. The Gerhart-Howden-Duran testing theory emerged in 1970 out of the desire by researchers to precisely define the notions of random testing and operational reliability (Hamlet, 2015). Hamlet (2015) explored the Gerhart-Howden-Duran testing theory to analyze persistent state-based testing methods that would increase the understanding of the statistical properties of the software. The Gerhart-Howden-Duran testing theory treats the behavior of software applications as nothing more than an input-output mapping. The research carried out by Hamlet (2015) showed that the Gehart-Howden-Duran testing theory critiques existing test methods. While there is no further research on this theory, I did not select this theory to drive the conceptual framework for this reason.

Software Reliability Testing Theory. The software reliability theory was considered for this study but was not chosen. The software reliability theory emerged during the early 1970s as an attempt by researchers to unify hardware and software applications for an overall reliable system (Lyu, 2002). While the software reliability theory supports the design of failure free software, the theory prohibits the estimation in advance of a project, and the amount of testing regarding execution time to achieve a specific goal. For this reason, the theory was not selected to drive the conceptual framework for this study.

Grey Systems Theory. In contrast to the testing theories, a researcher may consider using the grey systems theory. The founder of the grey systems theory was Professor Julong Deng in 1982 (Deng, 1982). As Huang and Wu (2018) noted, the grey systems theory is a quantitative method for dealing with known and unknown information. Moreover, the grey systems theory is studied using small samples and inadequate information (Sifeng, Tao, Xie, & Yang, 2016). So, the researcher must make assumptions and formulate conclusions based on incomplete information. The study conducted by Memon, Lee, and Mari (2015) indicated that the grey systems theory and the uncertainty theory are combined to achieve both quantitative and qualitative objectives. Since the quantitative method is not used in this study, I did not select this theory to drive the conceptual framework for this study.

Software Testing

Software testing is an entity of the software development lifecycle (SDLC) that has a plethora of research where various techniques were used. Evidence of this is shown by the number of original research investigations reported in the literature and those listed in the reference section. For example, Jacob and Prasanna (2016) published work proposing that software testing is an essential activity in the software development lifecycle. Julia et al. (2016) supported the workings of Jacob and Prasanna, adding that software testing is the most crucial phase in the software development lifecycle. Until now, there was a difference between software testing research and practice. The study conducted by Engstrom and Petersen (2015) explained that the reason for the gap is the discrepancy between how testing research is reported and how testing challenges are perceived in the IT industry. Garousi and Mantyla (2016a) argued that over 101 secondary studies had been published in software testing. Nevertheless, software testing is not an easy task; yet, it provides information about the quality of a product or service under test. When Julia et al. (2016) explained that it is essential to include the testing phase amongst all steps in the software development lifecycle, the goal was to improve the detection of software defects that may exist. The primary objective of the software testing phase is to confirm that the developed software product meets or exceeds customer requirements, is defect free, and ready for customer delivery. In any case, software testing ensures that expected business systems and software features behave as expected.

So far, software testing is a resource consuming activity. Per Zachariah (2015), a good fraction of software development cost is spent on software testing, since intensive testing is needed to identify and eradicate any future or potential software defects. Software testing is an activity that reduces software defects, and the goal is to deliver quality products at a low cost. Moreover, studies showed that testing constitutes more than 50% of the overall costs of software development (Afzal, Alone, Glocksien, & Torkar, 2016). Until now, software testing was an optional activity that was often implemented late in a project with little planning and executed carelessly (Anu, Hu, Carver, Walia, & Bradshaw, 2018). In hindsight, the result of a quick release may be the result of failure prone software. The first step to understanding what caused a software malfunction is to reproduce the defect that caused the failure through testing.

The desire organizations have for software projects to smoothly flow through the planning, analysis, design, testing, and implementation phases seamlessly with limited software defects is ideal. However, this is difficult because software development projects are immensely varied in their complexity and require a substantial amount of oversight and planning to be successful (Javed & Alenezi, 2016). Almugrin et al. (2016) found that in the software development process, well designed software is one of the most critical activities in the software development lifecycle; nevertheless, it is costly and not easy to test or maintain because of poor designs. In prior research, Fitzgerald and Stol (2017) explored the problem of severe disconnects between activities such as planning, testing, integration, and release providing a holistic view of the activities. Software testing is one of the most challenging labor-intensive practices of the software development lifecycle. The active support of software testing is essential to providing reliable software (Sun, Li, Leung, Li, & Li, 2015). Nevertheless, having quality software requires more than a dynamic process. Amanatidis and Chatzigeorgeou (2016) reported that an analysis of software evolution could reveal valuable information about software testing practices. So, the need for software to evolve for an extended period, as Kaur and Kaur (2015) noted, is necessary. In the end, evolution should not compromise the overall software quality by avoiding software testing.

This section of the study showed that software testing is an essential activity in the software development lifecycle. Software testing researchers have explained the reason for the discrepancy between how testing research is reported and how challenges are perceived in the IT industry. As Julia et al. (2016) pointed out, it is essential to include the testing phase amongst all steps in the software development lifecycle. While the goal is to improve the detection of software defects that may exist, some empirical studies suggest that software testing in some cases can be exhaustive (Afzal et al., 2016; Zachariah, 2015). When a defect occurs during preliminary testing, and the code is modified, the software may not function as expected. Although software testing is a labor-intensive practice of the software development lifecycle, discovering any defects in software is difficult and, for the same reason, complex. Therefore, testing boundary values are not sufficient to guarantee correctness.

Software testing is a technique that verifies and validates a product. Verification techniques can increase the effectiveness of testing (De Souza, De Almeida Falbo, & Vijaykuman, 2015). The software testing process is designed to verify and validate software. IEEE Standard for System, Software, & Hardware Verification & Validation determines whether the development product conforms to the requirements of the verification and validation lifecycle process (IEEE, 2017). Verification and validation are performed to help improve software quality. The purpose of verification is to confirm that the product is correctly built, challenging the requirements and design specifications (Sen, Marijan, & Gotlieb, 2018). When Tan et al. (2016) observed how test cases are used for verification, they noted that the test cases are chosen differently to avoid potential bias. The verification technique is a strategy used to implement static testing. The static testing strategy reviews software artifacts, including source code, while inspecting for defects without executing code during verification. It follows that validation techniques are challenging and controversial. The study conducted by Ahmed, Abdulsamad, and Potrus (2015) explained that the purpose of validation is to ensure that the product satisfied the quality standards set forth by the customer using the dynamic testing strategy. In contrast to static testing, dynamic testing is executed and is performed during validation. Thus, it is evident that software testing is crucial. Many studies have revealed the benefits of software testing and the importance of discovering software defects early on in the software development phase (Arora & Bhatia, 2018; Lemos et al., 2018; Lonetti & Marchetti, 2018).

For the last decade, the verification and validation (V&V) technique have been one of decreasing importance. Batarseh and Gonzalez (2015) explained that the reason for reduced concern is a result of the persistent software challenges and failures detected. Most researchers in the field support Ahmed et al. assertions that the only way to eliminate problems is by performing verification and validation. Validation is an essential component of the software development lifecycle as it provides answers to questions such as (a) Does the software fulfill its intended use? (b) Is the company building the right product? (c) Can the project be correctly implemented? (d) Are the documents in line with the development process? (Batarseh & Gonzalez, 2015). Thus, evaluating the effectiveness and efficiency of software quality is through verification and validation.

Software testing is a widely used practice for evaluating software qualities and assisting software developers with finding and removing software defects. Kirner and Haas (2014) found that software testing is an essential process that reduces the quality of software defects. Thus, the primary goal of software testing, according to Subramanian et al. (2017), is to ensure that a system or product fully satisfies all the requirements defined by the customer. The need for software testing is a critical part of software evolution because software defects can be expensive and dangerous (Kumar & Yadav, 2017). A case in point, Amazon's third-party retailers, were horrified when they noticed that items reduced to one pound as a result of a software defect. Then, a China Airlines Airbus A300 crashed because of a software defect that killed all 271 passengers (Mohan & Shrimali, 2017). So, software testing has a vital role in the software development lifecycle. Software testing is a process rather than a single activity that is primarily conducted to detect any or all errors that were induced in the system during various stages of the software development lifecycle (Petunova & Berzisa, 2017). The phases of the software development lifecycle include requirements, design, coding, testing, and maintenance (Mohammed, Niazi, Alshayeb, & Mahmood, 2017). Once software testing begins, requirements are collected, and the design and coding phases are complete. Garousi and Mantyla (2016a) pointed out that as new software is developed, it needs to undergo intensive testing to identify and remove potential faults or failures since the release of a quality software product is the ideal goal.

The key to assuring a successful and reliable software product or service is through intensive software testing. Garousi and Kucuk (2018) published work advocating the notion of software developers spending more time in the software development lifecycle to lessen the time spent in the software testing phase. Recently, the software testing community responded to research that the emphasis should be equally divided during the software development phase to address the time and cost incurred during testing (Bergmane et al., 2017). Zhou, Sinaga, Susilo, Zhao, and Cai (2018) concurred as they reported that software testing is a process that consumes about 30%-50% of software development time and budget. A typical testing fallacy, as explained by Dalal and Solanki (2018), is that software testing is merely an act of running test cases or running the software programs. The reality is that the actual test execution is part of the testing phase of the software development lifecycle. In 2015, Chen, Kuo, Towey, and Zhou reported that software testing is an approach that revealed software defects and problems as quickly as possible. Software testing activities start before the execution of test cases and continue even after the software testing phase is complete. The activities involved in software testing include test planning, selecting test conditions, creating and deciding on test cases, determining expected results, evaluating test results, evaluating the testing effort completion criteria, test status reporting, and finalizing the test phase (Rastogi, 2015). As explained earlier, verification techniques can increase the effectiveness of testing (De Souza et al., 2015). Therefore, the collection of verification techniques may be used during the development process to facilitate software quality. In summary, software testing activities can reveal design problems as well as operational and end-user issues. The advantage of early test planning and software development is that both force the software developer to think about the product from a testing perspective.

In this fast-paced age of technology, any newcomer or practitioner is likely to experience challenges in digesting large volumes of information about software testing. Most of all, documentation is an integral part of the software development lifecycle. Zhi et al. (2015), reported that a documentation's main usage includes maintenance support and program comprehension. For software testing, documentation aids in estimating the testing effort required, test coverage, and requirement traceability (Machado, McGregor, Cavalcanti, & Almeida, 2014). Some commonly used documented artifacts related to software testing include (a) test plan, (b) test design specifications, and (c) test case specifications (Steinberger, Reinhartz-Berger, & Tomer, 2018).

This section of the study discussed the importance of the verification and validation technique. Ahmed et al. pointed out that the only way to eliminate software defects is to perform verification and validation. As argued elsewhere, the importance of verification and validation is the flexibility of changes encountered during the software development lifecycle. Documentation is essential to software testing. A test plan is composed of detailed procedures that specify how and when testing finishes, who participates, and what test data would be used. Therefore, regardless of who creates the test plan, it serves as a guide to testing throughout software development. Although test design specifications record what needs to be tested, test cases are produced when the test design is complete (Cashman & Rosenblatt, 2014). Overall, testing is an important step when developing reliable and successful software applications.

Test documentation. The documentation for software testing provides support in estimating the testing effort required. Software documentation is an essential part of any software development process. However, according to Garousi et al. (2015), software practitioners are often concerned about the value, degree of usage, and usefulness of documentation during the software development lifecycle. The Institute of Electrical and Electronics Engineers (IEEE) published documents that establish specifications and procedures to confirm the reliability of the products and services used daily (IEEE, 2018). Specific to software and system testing, IEEE Standard 829-2008 determined whether the development products of a given activity conform to the requirements (IEEE, 2018). Swarts (2015) found that rich test documentation is easy to understand; whereas, poor test documentation is a hindrance for software developers to adopt when learning a new tool. Given Swarts (2015) research, Elberzhager, Munch, and Assman (2014) criticized the fact that fixing software defects is expensive and labor intensive because it is necessary to fix the defects not only in the software code but also in the documentation. As has been noted that software testing consumes about 30%-50% of software development time and budget; thus, the software testing activities must always be thoroughly documented to support resource allocation (Zhou et al., 2018). IEEE is a leading developer of International Standards that support many of today's information technology products and services. Specific to software testing, IEEE Standard 829 provided an outline for the format of artifacts used during software testing.

According to Phillips (2004), eight documents are specific to software testing, IEEE Standard 829-2008 (IEEE, 2018). The document types include test plan, test design specification, test case specification, test procedure specification, test item transmittal report, test log, test incident report, and test summary report. Although not all projects follow the full activities summarized in the testing process, most researchers are still uncertain of the testing process. I presented an illustration of a software testing plan in Figure 2 that contained eight document types included: test plan, test design specification, test case specification, test procedure specification, test item transmittal report, test log, test incident report, and test summary report.

Test analysis. The initial phase of the software testing process is the analysis phase. Hooda and Chhillar (2015) found that the test analysis phase encompasses the analysis of functional and nonfunctional requirements. Moreover, the requirements are to be clarified with the customers to identify the actual and expected results of testing and to identify any gaps. According to Ammann and Offutt (2016), test analysis is the process of examining something that could derive test information. Dolezel and Buchalcevova (2015) explained that the International Software Testing Qualification Board (ISTQB) is an international organization that provided standardized certification in the area of software testing. Therefore, the research carried out by Pawlak and Poniszewska-Maranda (2018) noted that the ISTQB process included planning, control, defining test conditions, designing, choosing test cases, executing test cases, evaluating results, evaluating exit criteria, reporting the process and the software under test and concluding the testing phase. Vukovic, Trninic, and Djurkovic (2018) argued that the goal of the testing process is to provide a basis for a testing process that is specific to testing business software in small and medium software organizations. Overall, the test analysis phase identifies what needs testing.

Test plan. The test plan is the general approach to testing or the design of the test. The test plan is the first document prepared that outlines the strategy that would be used to test a software application. The research carried out by Vasanthapriyan, Tian, Zhao, Xiongi, and Xiang (2017) noted that the goal of the test plan is to recommend the scope, approach, resources, and schedule of testing activities. Further, Vasanthapriyan et al. (2017) found that the advantage of using a test plan is that it forces software developers to think about the product from a testing perspective. Plus, they identified how the test plan should be structured. The test plan should include (a) an introduction to the test plan document, (b) assumptions made while testing the software, (c) a list of test cases for testing the software application, (d) a list of features to be tested, (e) the strategy to use while testing the software, (f) a list of deliverables needed for testing, (g) any risks involved during the testing process, and (h) a schedule of tasks and milestones to be achieved. In support of Vasanthapriyan et al. (2017) research, Wang, Wang, and Duan (2016) added that an optimal test plan is quite robust to the software testing phase. Hence, without a clear and robust test plan, a software developer might spend countless hours analyzing through test suites and fail to locate the problem. The test plan should be updated to indicate any divergence from the original plan.

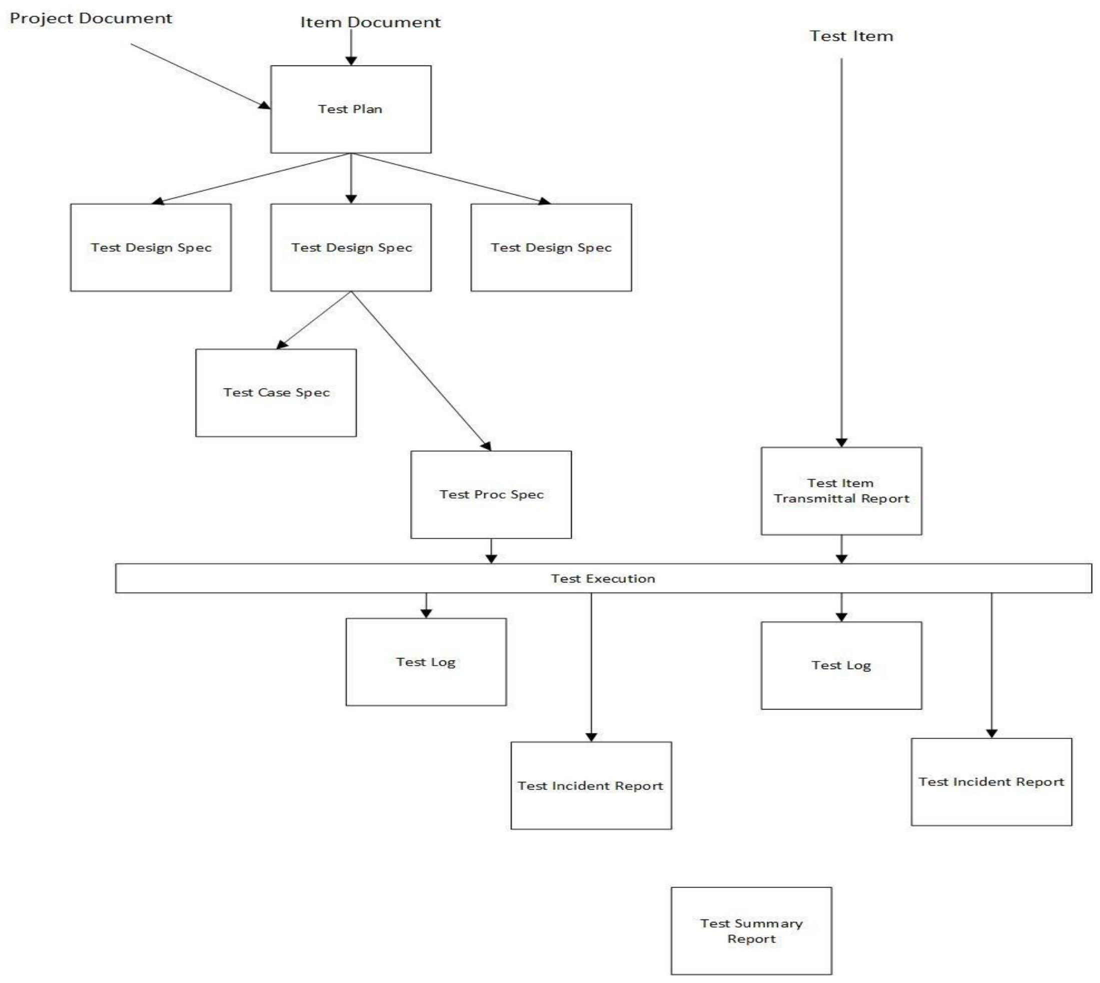


Figure 2. The hierarchy of test documentation. From "ISO/IEC/IEEE International Standard - Software and Systems Engineering -- Software Testing --Part 3: Test Documentation" p.10. Copyright 2013 by IEEE. Reprinted with permission (Appendix E)

Test design specifications. The test design specifications are the plan details or specifics for a test item and identify the associated test case. According to Vasanthapriyan et al. (2017), the test design is the first phase of developing test cases. The test design flows from the test plan to the software requirements specifications. While the test plan describes what must be tested, the test design describes how it should be tested, revealing design problems including, operational and end-user issues (Lemos et al., 2018). Thus, the document is used as a basis for the specification of test procedures and test cases. The test design specifications include (a) test design specification, (b) features to be tested, (c) testing refinements, (d) test identification, and (e) pass or fail criteria (Vasanthapriyan et al., 2017). The test design specifications record which structures should be tested and identifies how a successful test is recognized.

Test case specifications. Test cases are composed of a set of steps, conditions, and inputs that are used to validate tests. The research carried out by Sapna and Balakrishnan (2015) explained that test cases are designed from specifications represented by using the Unified Modeling Language (UML). One of the primary objectives of testing is to ensure that the changed system works correctly according to the written test case specifications. In 2018, Huber, Kuhm, and Sachse found that some test cases require days or even weeks to run. Later, the test cases are used as regression tests to ensure that the functionality of the previous code works. While waiting even a few minutes for test results can be detrimental to a software developer's workflow. Once all test cases have a passed status, the test is complete; otherwise, the testing phase starts over until the test case returns a passed status.

Consequently, test cases are vital because changes may introduce new software defects or unwanted side effects that must be avoided at all costs (Hooda & Chhillar, 2015). Although test cases are written to keep track of the testing coverage and to verify that code functions as expected; therefore, every test case should include (a) test case ID, (b) product module, (c) product version, (d) revision history, (e) purpose, (f) assumptions, (g) pre-conditions, (h) steps, (i) expected outcome, (j) actual outcome, (k) post conditions

(Gario, Andrews, & Hagerman, 2015). Since software developers design test case specifications, the ramifications are the test cases might contain software defects. To sum up, test case specifications are written to confirm that the software may function as expected.

Test procedure specification. The test procedure specification identifies useful information. Afzal et al. (2016) identified the test procedure as a deliverable product that flows from the test design specification. Thus, the purpose of the test procedure specification process is to specify the phases for executing a collection of test cases, or, more generally, the phases used to analyze a software product to evaluate a set of its features. According to the International Software Testing Qualifications Board (2018), test procedure specification is a document that examines one or more test procedures. In summary, the test procedure specification is used in conjunction with test case data to confirm the expected behavior of a test product.

Test execution. The phase of the software testing process that performs the test cases is test execution. The study conducted by Hooda and Chhillar (2015) determined that test execution begins when the criteria have been satisfied to avoid unnecessary delays in testing. Software testing researchers like Hooda and Chhillar indicated that whenever the actual and expected results do not match, the test is recorded as a software defect and assigned back to the software developer. However, there is a significant amount of completed work that focuses on the steps to report a valid software defect. Therefore, it is important to understand the software testing lifecycle to gain a thorough understanding of test execution.

Test transmittal report. The test transmittal report identifies test items submitted for testing, including the version and revision levels (Tuffley, 2011). According to Tuffley (2011), the transmittal report involves the person responsible for each item, location, and status. Moreover, any modifications made since the initial test analysis and test design specification phase are recorded in this report. Furthermore, the report documents the handover of the test items from the developer to the tester and confirms that the software product is ready for testing.

Test log. The test log produces a detailed chronological record of each step taken while performing the test. Tuffley (2011) noted that the test log is used by the tester to record the results of the testing. Furthermore, Tuffley explained that the test log verifies the number of defects identified while performing a process function. For Okoye, Naeem, and Islam (2017), the test log is used to evaluate and identify hidden defects, and the test cases are recorded as either passed or failed.

Test incident. The test incident report records any events that occur during the testing process that requires additional investigation and created during test execution. According to Tuffley (2011), the test incident report is used by the tester to document defects identified while testing and is used to initiate corrective action. Snyder, Zhang, Jasmin, Thankachan, and Donnelly (2018) indicated that the test incident report is generally a problem report. Overall, the test incident report is a summary of documented incidents, which may be the result of software defects.

Test closure. The test closure phase is essential. This phase ensures that all systems, integration, and user acceptance tests passed, and the summary reports are included (Hooda & Chhillar, 2015). The test closure phase provides a detailed analysis of software defects found or removed. The decision is taken whether all requirements are tested, and there is no critical software defect pending to be fixed or verified (Evans et al., 2018). Meanwhile, software testing researchers Hooda and Chhillar (2015) noted that once all the testing artifacts have been received and reviewed, then the software can be released. So, the test closure phase is the final step before the actual release.

Effective software testing is best achieved by using a structured and scientific methodology, instead of the historical break-it approach. When Ivanov et al. (2018) explained the goal of software testing, they ensured that software evolution does not break existing functionality. So, they concluded that few studies focus on methods that compare existing software concerning reliability. In 2016, Groce, Alipour, Chaoqiang, Yang, and Regehr reported that the goal of software testing is to improve software reliability and to reduce the risk of failure. Since the software testing phase is one of the last software development life cycle stages, the approach must be thorough and efficient, adding to the effectiveness and quality of the testing process (Jacob & Prasanna, 2016). After all, there is no fail-safe method of knowing whether tests are correct; to this end, the prevailing thought seems to have processes in place. The study conducted by Rais (2016) showed that the testing process could not be delayed until after the development phase; yet, testing should begin as soon as possible. A demonstration of this kind of implementation is the technique of test-driven development that includes writing the test case, executing the test case, and updating the source code. Rais (2016) explored that if the test continues to fail, there is a need to update the source code and retest it again.

Otherwise, the process continues until the design meets the requirements, and the test passes.

Similarly, software development and software testing are two distinctive, very well-connected phases within the software development lifecycle. Many studies have proven the benefits of testing and the importance of recognizing problems early in the development phase (Arora & Bhatia, 2018). Kirac, Aktemur, and Sozer (2018) concluded that the principle of software testing must be fast, and everything must work. Although software testing is not an easy task, Batool (2015) reported that the emphasis placed on the importance of testing could lead to many problems if not correctly detected. The ramifications may lead to late deliveries, over budgeting, or failure to deliver the required features. When Zalewski and Gonzalez (2017) talked about software defects, the most prominent example they provided is the case of NASA's 1997 Pathfinder mission to Mars where a software glitch impacted the real-time kernel of the rover control software calling for an in-depth analysis back on Earth at the mission control center in Jet Propulsion Laboratory (JPL). After repairing the glitch, the software was tested and uploaded back to the rover on Mars.

As software evolves, updating the required specifications is a manual task that is expensive and time consuming. Consequently, researchers and practitioners have expressed concerns that software testing demands a large share of the costs of a software project (Lemos et al., 2018). For example, a 2015 survey conducted by the Capgemini Group (2016) revealed that 35% of the spending budget was allocated to software testing practices. Altogether, software testing is a tradeoff between budget, time, and quality.

For these reasons, many software testing strategies and techniques are used for verification and validation of software, which I discuss in the next section. Further, the study provided a mechanism to show how the testing strategies contributed to the overall effectiveness of software testing.

Software Testing Strategies

Software testing is an activity intended to evaluate an attribute or capability of a program or system with the determination that it meets the required behavior. Despite the importance of software testing, it involves exploring the behavior of a product to discover potential faults (Barr, Harman, McMinn, Shahbaz, & Yoo, 2015). Barr et al. found that much work on software testing seeks to automate as much of the testing process as possible, allowing for testing to be faster, cheaper, and more reliable. There are two methods of software testing, including manual testing and automation testing. In manual testing, the responsibilities of test planning, test execution, and documenting software defects are manually performed by human efforts (Mohan & Shrimali, 2017). Although software testing is the first approach; yet, it requires intensive manual efforts. In 2015, Chen et al. reported that many software releases were scary experiences because the release process was not practiced. Instead, many error prone manual activities were to blame. They are further adding that the setup and configuration of the test environment contributed to the blame of errors for three weeks.

In automation testing, test scripts are designed for beginning the testing and execution of a product. The technique takes less time, requires higher accuracy, and is more expensive than manual testing (Mohan & Shrimali, 2017). Most organizations associated with software testing with automation as a solution to decrease testing costs and reduce cycle time in software development. Since automation testing is quite comprehensive, manual testing is often a necessity. Many software testing strategies can be implemented either as a manual or automation testing process. Xiao, Liu, and Wang (2018) defined a software testing strategy as a framework that identified the testing approach of the software development lifecycle made to inform software developers, project managers, and other practitioners of some major issues detected during the testing process. Further, Xiao et al. (2018) explained that a software testing strategy helps to manage a test suite by identifying redundant test cases. Some commonly used software testing strategies include (a) unit testing, (b) integration testing, and (c) system testing.

This section of the study explored the two methods of software testing: manual and automatic. As Mohan and Shrimali (2017) pointed out, manual testing is performed manually by human efforts, and automation testing is performed by the assistance of tools, scripts, and software. As argued elsewhere, the limitation of manual testing is that it requires an exponential amount of time and human effort, just as automated testing is more efficient and requires a more significant investment in tools (Chen et al., 2015). Though many software testing strategies could be implemented as either automatic or manual, this study discussed the three most common testing methods that software developers use to ensure the reliability of software applications. The methods included unit testing, integration testing, and system testing. Further, the various software testing strategies available to software developers are discussed in the next section.

Unit testing. A unit test is a testing method that gives the ability to verify that functions or small units of code work as expected and tend to mirror the operational environment. Evans et al. (2018) indicated that a unit test ensures that a software product is defect free; whereas, a well-designed software application has minor software defects and high cohesion. An advantage of the unit testing strategy, according to Eler, Endo, and Durelli (2016), is that the developed code is easy to test and prevents future code changes from breaking the functionality. While it is true that unit testing is the lowest level of software testing, Hooda and Chhillar (2015) identified that software developers usually conduct unit testing. Although this may be true, Khan (2016) demonstrated that unit testing is difficult and time consuming. Even today, software testing strategies are confined to the unit testing realm before progressing onward to integration testing. There are two distinct categories of unit testing: black-box testing and white-box testing. Black-box testing is conducted independently of the software implementation; whereas, the implementation drives white-box (or glass box) testing.

Black box testing. Black box testing is a software testing strategy designed to adequately exercise the functional requirement of a system without regard to the fundamental workings of a software product. The black-box approach design functional test cases to include all functionality to be delivered. The most widely discussed black box testing strategies, according to Jan, Shah, Johar, Shah, and Khan (2016), include (a) equivalence testing, and (b) boundary value analysis. Black box testing has become harder and urgent; however, research has revealed impressive results addressing many of the aspects of the problem that spans from integration with development to test case generation and execution (Henard, Papadakis, Harman, Jia, & LeTraon, 2016). When Mariani, Pezze, and Zuddas (2015) talked about black-box testing, they did not mean an input or output driven approach to ensure that the functionality performed as specified, but they suggested that all parts, not some, of the back end of the code, had been tested. The equivalence partitioning approach determines the subset of test cases. For a detailed discussion, the research carried out by Lemos et al. (2018) separated the input-output domains of a software application into equivalence classes, whether valid or invalid. The boundary value analysis approach converges on the boundary conditions of input and output equivalence classes. Experience revealed that test cases that examine boundary conditions have a tremendous payoff than test cases that do not (Burman, Hansbo, & Larson, 2018). Thus, black-box testing ensures that the software code will function according to the specifications.

White box testing. White box testing is a software testing strategy that focuses on the internal logic and structure of the code. Software testing can never completely identify all the defects detected in the software. Zhou et al. (2018) identified white box testing as the best technique for code optimization. As a result, their study concluded that the test strategy used achieved considerable savings in comparison to the number of test executions required to detect software defects. In like manner, Larrea (2017) identified the most widely used white-box testing strategies to include (a) fault-based testing, (b) statement coverage, (c) branch coverage, (d) condition coverage, and (e) path coverage. For example, when Emam and Miller (2015) talked about statement coverage, they ensured that every statement in a program executed at least once. In another example, Yi, Tan, Mechataev, Bohme, and Roychoudhury (2018) reported that statement coverage was among the first testing strategies invented for white box testing. Even though the complexity and size of the software are growing, branch coverage aimed to detect and correct some of the software defects to ensure reliability. The research conducted by Godboley, Panda, Dutta, and Mohapatra (2017) proved that any testing strategy that generates enough test cases to execute produces 100% branch coverage.

In contrast, Manikumar, Keumar, and Maruthamuth (2016) indicated that it is unlikely to generate a set of test cases to verify for 100% defect free code. Meanwhile, Schwartz, Puckett, Meng, and Gay (2018) conducted a preliminary study considering code metrics as a feature. Despite the severity of the features observed, the results of the investigation revealed that using machine learning to predict branch coverage using automated testing is viable; yet, a feasible option.

The condition coverage technique encompasses a variety of requirements. Condition coverage, according to Kandl and Chandrashekar (2015), involved the evaluation of the testing process of software incorporating decisions that contain multiple boolean expressions. For instance, the decision (AB) test cases (CD) and (DC) meet the coverage criterion but does not cause decisions to take on all possible outcomes. As the main requirement of condition coverage, all boolean assignments must adequately define the input variables. Khari Lumar, Burgos, and Crespo (2017) proposed an approach that provided a set of minimal test cases with maximum path coverage in comparison to other software testing strategies. Their proposed study generated optimal test results used for automated fault detection. Moreover, Goel and Mehtre (2015) analyzed white box testing, and the strategies involved noting that white box testing requires a deep understanding of the testing network or system providing better results. In the end, their study showed that white box testing is time consuming and exhaustive.

Integration testing. Following unit testing, the next level of testing is integration testing. So far, integration testing is becoming more critical because of the increased focus on modularity and abstraction. Earlier, software testing researchers like Garousi, Felderer, Karapicak, and Yilmaz (2018b) indicated that integration testing is performed to test the functionality of grouped modules. This type of testing is performed between unit testing and system testing to test functionally grouped components. Integration testing is a method that determines whether independently developed modules of software work when joined and tested as a group. Integration testing also exposes errors in the interaction between integrated modules. The goal of this testing strategy is to test the interface between modules and units. The research carried out by Shin and Lim (2018) identified integration testing based on previously tested modules or units as an advantage. The modules are tested separately, and testing is done by integrating already tested modules. However, according to Milajic, Beljakovic, Davidovic, Vatin, and Murgul (2015), integration testing is challenging to debug, and much throwaway coding is required. Integration testing uses a pattern approach to validate software. Sadath, Karim, and Gill (2018) suggested that integration testing is just like extreme programming (XP) testing techniques despite the fact of exploiting the agile experimental methods. XP programming uses a simple methodology that executes smaller deliverables when designing and testing code (Sohaib, Solanki, Dhaliwa, Hussain, & Asif, 2018). The level of integration testing verifies the structure of the software program by examining the software application's interface. Moreover, integration testing starts when the software code matures and proceeds until the software developers release the product to the next phase.

The software testing strategies associated with integration testing differ from traditional software. Larrea (2017) identified the most widely used integration testing strategies to include (a) big bang, (b) top-down, (c) bottom-up, and (d) mixed. When Lonetti and Marchetti (2018) talked about the big bang, attention was not given to verifying the interfaces across individual units. Instead, the components are linked together and tested all at once. The evidence exhibited by the wealth of testing research investigations revealed a lot of the techniques and tools available for integration testing; despite the difficulty to implement. For example, Chen, Wu, Lin, and Ye (2018) found that the top-down testing strategy is used to simulate the behavior of the lower-level modules that are not yet integrated. Their research further explored both the bottom-up and mixed testing strategies. The results of the research revealed that the bottom-up testing strategy test units at a lower level with the help of higher-level units; whereas, the mixed testing strategy combines both top-down and bottom-up approaches. To simplify, the strategies identified for integration testing are very complex, and testing may take hours, days, or even weeks to complete.

System testing. The final level of software testing strategies discussed in this section of system testing. System testing projects the big picture to ensure that the entire system is functioning correctly and assumes the responsibility by evaluating the quality of the software under test. Suffian, Fahrurazi, Ann, Aman, and Bajuri (2018) found that system testing helps to reduce the risk of failure when software operates in its intended environment. After all, Khan and Amjad (2016) acknowledged that system testing is to be used to test the system. For this reason, the testing team conducts system testing. Larrea (2017) identified the commonly used system testing strategies to include (a) alpha testing, (b) beta testing, (c) acceptance testing, (d) regression testing, (e) performance testing, (f) volume testing, and (g) stress testing. The illustration in Figure 3 showed the most common types of software testing strategies.

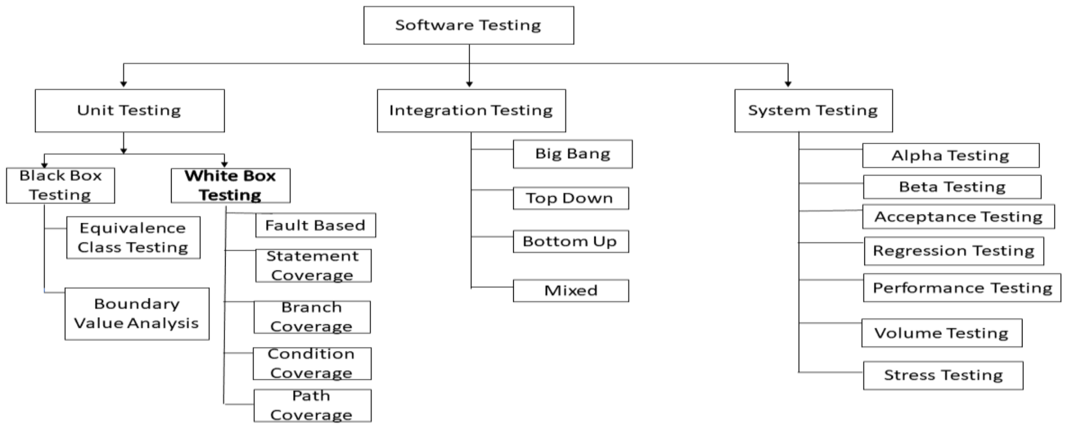


Figure 3. Classification of Software Testing. From "A Comparative Analysis On Blackbox Testing Strategies," by P. M. Jacob and M. Prasanna, 2016, *2016 International Conference on Information Science (ICIS)*, p 2. Copyright 2016 by IEEE. Reprinted with permission (Appendix F).

As an example, Jamil, Arif, Abubakar, and Ahmad (2016) published work proposing that inhouse software developers perform alpha testing in real-life environments. Furthermore, alpha testing promoted the quality of the product by exposing errors and common user issues before deployment. When Lonetti and Marchetti (2018) talked about alpha testing, there was no test plan to follow, but the system was deployed to end-users, and testing could not perform without the involvement of the software development team. The product was released to specific end-users for the testing phase. Stavova, Dedkova, Ukrop, and Matyas (2018) suggested that beta testers should represent a future product's end-user as much as possible. The beta testing strategy does not require unique testing environments, and the focus is not placed on deployment and workload issues. Furthermore, Saeed, Khan, Khan, and Islam (2018) found that beta testing measures the satisfaction of end-users in contact with the software product. The beta testing methods have made it possible to evaluate the design and usability of a software product.

Acceptance testing is a testing strategy that is utilized for customized software and applications. Customized software applications are designed for the usage of internal business sectors or a select group of end-users (Fylaktopoulos et al., 2018). During this stage of testing, the end-user collaborates with the software developer to verify the software requirements specified in the statement of work. Leotta et al. (2018) presented research explaining that the goal of acceptance testing is to assess the system’s compliance with the business requirements and to verify if it has met the required criteria for delivery to end-users. Then, stakeholders must sign off on the process once users agree that the software is functioning as designed. Hence, acceptance testing is the final approval process in customized applications and is more effective when testing on a larger scale.

Software testing is the most common industry practice to validate the correctness of software. Software developers often write test cases for recently implemented code while checking the functionality and adding these tests to a test suite. Moreover, to check that software modifications did not break previously working features, software developers practice the technique of regression testing, which is running test cases at each software revision. Even though regression testing is necessary, it is expensive because of the many numbers of tests executed. Some studies revealed that regression testing could take up to 50% of the testing budget (Hamill & Goseva-Popstojanova, 2017). In 2017, a large software company revealed that software failures caused $1.7 trillion globally in financial losses (Garousi, Ozkan, & Betin-Can, 2018). Motivated by a need to improve regression testing practices, Garousi, Felderer, Karapicak, and Yilmaz (2018a) proposed an approach that yields more efficient test suites in comparison to the traditional manual test selection approach. The research revealed that the suggested practices have been beneficial in saving the costs of regression testing. Earlier Parsons et al. (2014) explained the central role of regression testing for maintaining quality software. Hence, the study revealed that investing in regression testing tools and practices is likely to be beneficial for organizations. Although regression testing is a critical component of software development and maintenance, empirical studies indicated that using functional testing or structural testing alone cannot detect all software defects detected in a software application (Parsons et al., 2014). Since tests in a test plan depend on a chosen testing strategy, the testing strategy that is used by other testing techniques should be the same if the regression testing process involves the reuse of existing tests. Therefore, software developers may use the regression testing approach to ensure the reliability of software applications.

Performance testing is an effective way to measure the system parameters in terms of response time and service availability. The research carried out by Sanchez, Delgado-Perez, Segura, and Medina-Bulo (2018) explored the availability of mutation testing to assess the improvement of the performance testing strategy. Examples and open-ended questions motivated the study. Sanchez et al. (2018) proposed the generation of real defects seeking not to alter the semantics of the program. The outcome of the investigation concluded that previous research challenges need to be resolved such that they are crucial to enhancing the ability of tests to reveal performance software defects. The research carried out by Ahmad, Truscan, and Porres (2018) introduced an approach for testing applications in which they identified the worst path in a workload model, causing the highest consumption of a given resource. Hence, the study revealed that in the case models with a significant amount of data, the approximate method performs better. Although performance testing is a significant activity to ensure quality in continuous software development environments, performance is all the more of what people care about (Sanchez et al., 2018). To this end, performance testing has been a significant concern, and the driving force of software evolution (Ahmad et al., 2018). Thus, the overall goal of performance testing is to identify the performance bottleneck of a typical software system.

Volume testing is a nonfunctional testing strategy that performs performance testing techniques. The test data is generated using test data generation tools. For a detailed discussion, the research carried out by Nichita (2018) verified the ability to manage vast amounts of data either as input or output that resides within a database. The study concluded that with proper scaling, iterations are robust and convey swiftly for most conditions. The final testing strategy used during system testing is stress testing. Stress testing is performed only by software developers. The strategy encompasses a set of executables used to simulate or stimulate abnormal behavior detected in a software application (Di Alesio, Briand, Nejati, & Gotlieb, 2015). The purpose of stress testing is to consider situations that generally shut down or produce changing conditions in a software application. Thus, testing explicit constructs in a software program exposes vulnerabilities in the software. Stress testing commences after the software coding phase is complete and proceeds until benchmark results are satisfied. The stress testing strategy, alone, validates the stability of the application, but it does not provide data for post development objectives.

This section of the study explored the various types of software testing strategies software developers use to ensure the reliability of software applications. Empirical studies have demonstrated that there are many approaches to improving software testing (Evans et al., 2018; Hooda & Chhillar, 2015). Software testing is challenging, as Mariani et al. (2015) pointed out, whereas some software developers believe that new software technologies are needed (Lemos et al., 2018). Thus, developing new software testing technologies is risky. The waterfall and agile methodologies determine what developing techniques, professional skills, necessary tools, and management actions are required to improve the quality of software testing. The details of the waterfall methodology and agile methodology are discussed in the next section.

The Waterfall Methodology

In this section, I introduced the fundamentals of the waterfall methodology and provided an understanding of its purpose for software testing. The waterfall software development methodology is a classic, tried and true method that has proven to be beneficial over the years. Understanding the difference between method and methodology is of great importance. According to Chen and Han (2018), a method is a research tool to generate and analyze data, while methodology is the reason for using a particular research method. For many years, there have been significant debates about whether "methodology" or "method" is the correct term, what constitutes a methodology, and how practices differ from methodologies (Gupta, 2018). In this study, I used the term methodology. The meaning is consistent in the perspective that a development methodology emerges out of a philosophical view. Now, the understanding that any methodology is more than the sum of its practices is key to understanding why traditional methodologies differ from agile methodologies. Because of the importance of understanding the philosophy and motivation behind any methodology's practices, waterfall and agile are leveraged to ensure the success of a project. Many researchers have focused on engineering practices of specific agile development methodology. In contrast, this study focused on software testing differentiators between the waterfall and agile methodologies.

The waterfall methodology is a traditional methodology that has demonstrated to be effective over the years. The methodology focuses on top-down development, beginning at the highest level and efficiently narrowing the scope and design down until the lowest level of detail is reached (Kramer, 2018). Projects are managed more efficiently when segmented into a hierarchy of system requirements, constraints, exceptions, and feasibility. The waterfall methodology assumes that the “facts” regarding the system are available upfront with relative accuracy and certainty. In their research, Chari and Agrawal (2018) reported that work is completed in stages while content reviews are conducted between stages; reviews represent quality gates and decision points before proceeding. According to Heeager and Nielsen (2018), the waterfall methodology provides sequential steps and ensure the adequacy of documentation and design reviews to enhance the quality, reliability, and maintainability of the developed software. The waterfall methodology is used to get quick fixes out to end-users.

Upon completion of the design phase, the coding and debugging phases commence. During these phases, the software development team builds the product. The team then performs unit testing and integration testing related activities. Once testing starts, it is difficult to go back. After a while, the completed product is delivered to the testing environment. When Chari and Agrawal (2018) talked about the waterfall methodology, their published research suggested advantages over the previous, ad hoc model for development. First, formal requirements and design procedures are established, allowing for better code quality and end-user acceptance. Second, the recognition of formal testing phases is needed since the waterfall methodology relies on the creation of full documentation requirements during the early stages of the project. As Adnan and Ritzhaupt (2018) pointed out, the waterfall methodology is effective for some classes of software even when projects are short. In contrast, the waterfall methodology is not as successful for complex and interactive end-user focused systems. Contrary to research, the waterfall methodology is a poor model for long and ongoing projects. In general, it follows that the software development team can design software from the documented requirements.

Software testing requires an understanding of what and how to test. Therefore, software developers should avoid making late revisions since the focus is placed on early definition and requirements gathering. The waterfall methodology assumes a linear delivery model, with the successful completion of each phase, although later views of the waterfall model acknowledged feedback loops between adjoining phases (Chari & Agrawal, 2018). While the classic waterfall methodology is not suitable for handling changing or uncertain requirements, organizations spend as much as 40%-50% of their budgets on requirements elicitation, analysis, and design (Primiero & Raimondi, 2015). On the one hand, software developers assumed that too often end-users changed their minds, on the other end-users criticized that it is the software developers who do not provide an accurate understanding of what the system would deliver. Figure 4 illustrates that the cost of change becomes significantly higher as time progresses, according to Boehm (2002).

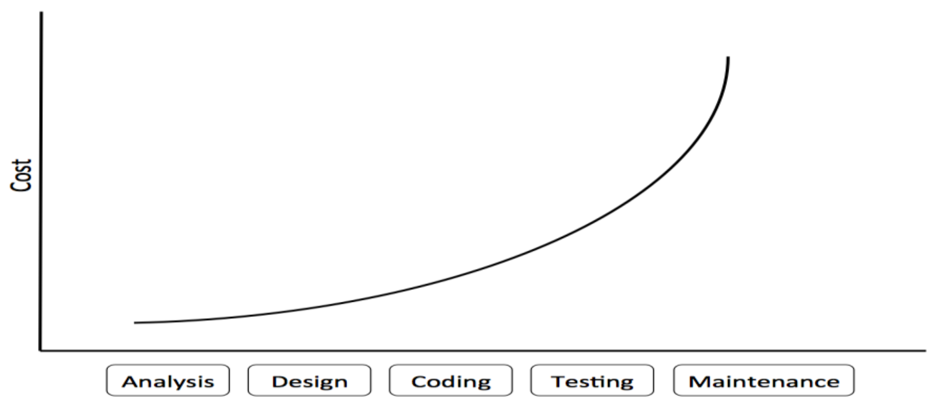


Figure 4. The cost curve of change. Adapted From "Get Ready For Agile Methods, With Care" by B. Boehm, 2002, p.68. Copyright 2002 by IEEE. (Appendix G)

Therefore, the software testing phase is prolonged, and the timescale of project delivery under the waterfall methodology is usually measured in months, and often years. There is no question that projects tend to escalate. For example, Garousi and Kucuk (2018) published work suggested that managers need to justify their decisions, which may carry a psychological impact because of sunk costs. To that end, the traditional waterfall methodology delivers most of the value at the end of the project to include software testing, and consequently, there is a long delay before a value is delivered and before progress and success can be measured. Thus, pressure escalates further to deliver the product.

The Agile Methodology

The agile software development is prevalent and has attracted an enormous following, even an entire community of users. During the mid-1990s, software practitioners began to acknowledge the problems that resulted from strict adherence to past traditional approaches (Matharu, Mishra, Singh, & Upadhyay, 2015). As practitioners started to make a note of the best practices from lightweight iterative methodologies, the result transitioned into the Agile movement. Moran (2015) identified the dynamic restructuring of the software development lifecycle as the silver bullet for increased productivity. After all, the software development process is well suited to pander to the synopsis of continually changing or evolving requirements.

Agile uses a spiral model, which depicts a sequence of iterations, or revisions based upon user feedback. In other words, agile takes a get something started approach to build a product that involves testing and revision practices (Nidagundi & Novickis, 2017). Agile is a discipline of the software development lifecycle based on the values of adaptability, communication, and feedback. Despa (2015) indicated that the primary focus of the agile methodology correlates with adaptability and communication. Even now, the methodology requires intense interaction between software developers and end-users. Once the customer and software developer agreed to a list of tasks for each cycle of iterations, changing requirements became challenging.

Agile brings the entire software development team together. Although it is a lighter, more people centric approach in comparison to traditional waterfall approaches, the process is simple and delivers software in quicker timeframes soliciting feedback (Lei, Ganj-eizadeh, Jayachandran, & Ozcan., 2017). According to Huckabee (2015), agile lifecycles are geared toward the delivery of working software. For this reason, when using the agile methodology, the emphasis is placed on constant feedback, and each incremental step is affected by the determined actions of the preceding step. Since the agile process determines the result, critics often claim that the quick iterations and fast releases lack discipline and produces products of questionable quality. In recent years, Odzaly, Greer, and Stewart (2018) criticized the agile process as being overly process centric.

Similarly, Despa (2015) and Joann (2015) found that agile methodologies do not scale well in larger organizations because design issues detected during testing are expensive and difficult to correct. When software developers write their tests, they are more vested in fixing them when they fail. Even though Cooper and Sommer (2018) noted that less focus on testing would be the expectation, the overall severity of the agile software development team practices remains high. So far, the agile methodology has resulted in many management methods for software development projects and practices that shift from a cumbersome process to lightweight methods. The literature published by Tripp and Armstrong (2018) identified at least eight unique agile frameworks: (a) adaptive software development (ASD), (b) crystal, (c) dynamic system development method (DSDM), (d) extreme programming (XP), (e) feature driven development (FDD), (f) lean software development (LD) (h) scrum, with scrum and XP being the most frequently implemented. Their study examined how a fit between an organization's goals for agile adoption may affect a project's performance. In sum, as agile frameworks become popular, many organizations are proposing to implement packages that software developers can use to manage and document the agile process effectively.

Comparing Agile and Waterfall Methodologies

In agile, software developers test their work before revealing to the rest of the project. The observance of quality testing is a significant benefit of the agile methodology. Historically, software testing has been executed manually and identified as error prone, time consuming, and expensive (Kour & Singh, 2016). Just as the software industry extensively adopted the agile methodology, Alahyari, Svensson, and Gorschek (2017) reported that agile is a familiar example of a lifecycle used to build intelligent and analytical systems. Their study focused on the delivery of valuable software. The agile methodology involves the use of multiple sprints.

Moreover, each sprint has a specific software feature to develop, test, refine, and document. Since agile depends on the context of the project, testing is performed differently for every sprint. After all, software testing researchers like Vijayasarathy and Butler (2016) supported the agile methodology; whereas, Campanelli, Camilo, and Parreiras (2018) criticized the model arguing that all organizations are not ready to fully transition to agile. The authors presented results that showed that external environment criteria influenced the adoption of agile practices. Most notably, a recent study by Kour and Singh revealed that Lehman's law of declining quality supports the agile software development methodology. Furthermore, the agile methodology is in alignment with the feedback system proposed by Lehman (1996), as well as the law of continuing change (Lehman, 1996). Agile has had a higher success rate than any other software methodology despite the challenges encountered.

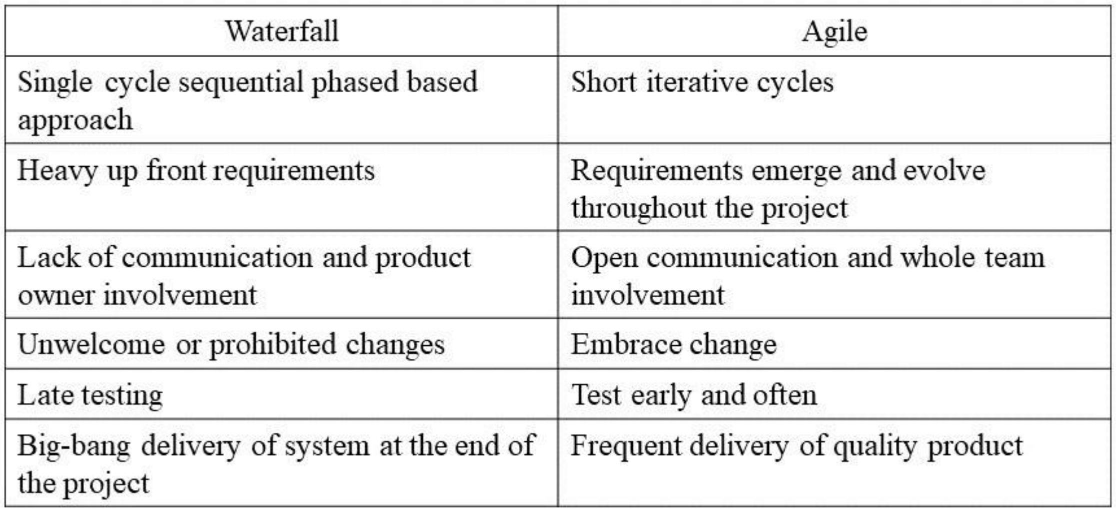
There are two contemporary software development methodologies in use, waterfall and agile, besides each model carries its issues. Despite the widespread use of the waterfall lifecycle, there are numerous software defects. For example, Poth (2016) reminded us that it is based on paper, and the requirements are gathered and finalized early in the process before development starts. Then, the software is delivered as a finished product before the stakeholder has an opportunity to use it. A software project based on the waterfall model is prone to a delay in the detection of errors. Finally, both software reuse and prototyping are not formally tested according to the waterfall methodology. Software reuse has value providing an understanding of how software applications work. While software reuse saves time and cost, empirical studies reveal that it may redefine software maintenance (Martin, 2017; Poth, 2016). Although ad hoc changes are inexpensive and applied swiftly, they are likely to degrade the software structure; instead, planned changes preserve the software structure (Kramer, 2018).

Large-scale testing is a smart way to test. Nowadays, companies are moving away from formal methods of testing to large-scale testing in which components are comprised to identify weaknesses in the software application or service (Alvaro & Tymon, 2017). In the waterfall method, committee members and project sponsors are required to sign off at the end of each phase. Steinke, Al-Deen, and LaBrie (2017) noted that the intention is not to move forward until the design phase is complete. Thus, as a downfall to the waterfall method, software developers are overwhelmed with satisfying project approvals and meeting deadlines such that focus is lost on developing a reliable product on time and within budget. Some researchers consider the waterfall method to be an old or outdated methodology in comparison to agile for all that it is worth; it is still popular (Politowski, Fontoura, Petrillo, & Gueheneuc, 2018).

The challenge all companies face in a swiftly changing business environment is competitive. In most companies, software testing practices and processes are far from being mature, and they are usually conducted in ad hoc fashions (Garousi, Felderer, & Hacaloğlu, 2017). The primary focus when using agile is to achieve customer satisfaction. When Cruzes, Moe, and Dybå (2016) talked about agile, they explained that agile is a methodology used to develop a system or product incrementally by building continuous prototypes and adjusting to user requirements. One of the underlying problems in software development is the difficulty customers have in explaining their needs. According to Inayat, Salim, Marczak, Daneva, and Shamshirband (2015), customers faced challenges when explaining their requirements. Agile software development helps customers to define their requirements, and it has led to many successful software development projects. When Martin (2017) indicated that agile is a software development methodology that emphasizes adaptability in a collaborative process, software developers were ready to deliver tangible products in short iterative cycles. In the end, testing and feedback are continuous such that software defects or requirement changes can be discovered, clarified, and addressed throughout the development process.

Once agile development was believed to best suit small teams, but the success has since inspired the use in large-scale development environments. The research carried out by Dikert, Paasivaara, and Lassenius (2016) identified problems of large-scale software development environments. Their research revealed that preexisting development environments are incompetent for supporting large-scale software systems. Furthermore, their work provided insights into future work. Most notably, the recommendation that a model needs to be defined, specifying the attributes of the software testing environment.

The waterfall methodology is used widely in both large companies and government projects. The agile methodology embraces change and emphasizes open communication and whole team involvement. Moreover, Curcio, Navarro, Malucelli, and Reineher (2018) found that the waterfall process is developed and executed in sequential order. When Raschke et al. (2015) talked about the agile process, they explained that testing occurs early and often. After all, agile testing requires software developers to manage changes quickly using proper tools, without compromising safety and quality. As illustrated in Figure 5, the waterfall methodology emphasizes heavy up-front requirements.



Waterfall	Agile
Single cycle sequential phased based approach	Short iterative cycles
Heavy up front requirements	Requirements emerge and evolve throughout the project
Lack of communication and product owner involvement	Open communication and whole team involvement
Unwelcome or prohibited changes	Embrace change
Late testing	Test early and often
Big-bang delivery of system at the end of the project	Frequent delivery of quality product

Figure 5. Waterfall versus Agile. From "An Analysis of the Software Selection Process Using Waterfall Versus Agile Methodologies: A Simulation Study" by S. Feddock, 2016, p.23. Copyright 2016 by Proquest. Reprinted with permission (Appendix H)

Software Testing and The Federal Government

The secret to 21st century software success is innovation. In this era, companies in every industry continue to transition in the direction of software innovation. Moreover, software innovation has captured the attention of all the sectors, including accounting, banking, education, healthcare, and even the United States federal government (Anand, Singh, & Das, 2015). Since the early 1970s, the waterfall methodology has been the dominant approach for software development (Dolezel & Buchalcevova, 2015). The study conducted by Ashmore, Townsend, Demarie, and Mennecke (2018) credited Winston R. Royce as the founder of the waterfall methodology who described the process as a cascading set of project phases that include requirements, analysis, design, code, test, and operations. Software testing is difficult work. As reported by Ashmore et al., Royce criticized the waterfall methodology as a flawed approach because existing testing tools are considered inadequate. Thus, to understand why the waterfall methodology failed government IT projects, I explored the healthcare.gov website.

During October 2013, the Centers for Medicare and Medicaid Services (CMS), an industry sector within the Department of Health and Human Services (HHS), initiated the healthcare.gov website under the nation's Affordable Care Act health reform law (Capili, 2018). The law gave numerous Americans the stability and flexibility to make informed choices regarding their healthcare. The healthcare.gov website was a data repository configured to allow citizens the opportunity to choose their healthcare policy (Capili, 2018). According to Huang (2014), the healthcare.gov was another federal government IT project that encountered catastrophic failures while using the waterfall method. Consequently, the website suffered from poor planning from the very beginning and the lack of software testing. Again, the importance of software testing should be unavoidable. Instead of using an agile approach that would allow for the release of segments of the system in weekly or bi weekly sprints, software developers used a big bang approach, whereby all the components were tested at once until a finished product was released (Anthopoulos, Reddick, Giannakidou, & Mavridis, 2016). Once the website launched, access was granted to residents in 36 states to create and manage their healthcare exchange. After the website went live, the discovery was that it was not designed to support the large volume of end-users, such that within a 2-hour timeframe, the system crashed (Cundiff, McCallum, Rich, Truax, & Ward, 2015).

The implementation was a massive IT failure. As a result, the failed launch had to be rescued by a team of Google software developers in an emergency turnaround that cost $100 million over the initial budget (Mergel, 2016). The U.S. Government Accountability Office (GAO) (2014) report concluded that the lack of understanding implications, and the frequent changing of requirements while pursuing a compressed timeline to release the software were significant factors that contributed to the performance of the website. Further, the U.S. Government Accountability Office (GAO, 2014) report revealed the inefficient management of project expenditures even though an increase in funds occurred during the development process. Thus, failure to understand that software rework and poor-quality software development impacted the agile schedule causing for flawed customer requirements. In sum, it was not until 2014 that the website became fully functional.

The federal government is looking to adopt the use of agile to quickly deliver innovative software that satisfies the needs of the customer. So far, federal government agencies are facing IT upgrades and legacy issues, such that outdated systems and acquisition processes are the results of high-risk technology projects that are over budget and behind schedule (Misra, Bisui, & Mahapatra, 2018). The agile software development methodology follows the mantra fail fast, early, and often instead of failing catastrophically and wasting taxpayer dollars as observed with the healthcare marketplace portal (Mergel, 2016). In 2017, Deborah Sills, Kevin Tunks, and John O'Leary published a study on the California Health and Human Services agency. For years, work on a traditional waterfall request for proposal (RFP) was in progress, until a sudden move was ordered to switch to agile. According to Stuard Drown (as cited in Sills et al., 2017), the release of an RFP for a $500 million project in 2015 was in progress. Although the project was on version seven, the work had been performed for three years and near completion. Thus, in a dynamic switch, organizational leaders determined to transition the project to agile. The move caused the agency to break down the project modules and employ multiple vendors. Ordinarily, innovation in government software development is created by using an agile software development approach adopted from the private sector and IT organizations.

This section concluded the literature review. By utilizing Lehman's laws of software evolution as the underlying conceptual framework, it provided a unique lens by which to view testing strategies and the perceived benefits for implementing these strategies within this case study. The review of the literature focused on software evolution, software testing, and the testing strategies software developers used to ensure reliable software applications in the government contracting industry.

Software testing is a crucial phase of the software development lifecycle (Lemos et al., 2018). Since the first piece of written software code, there has been a need to test code to ensure that it functions appropriately (Alvaro & Tymon, 2017). Historically, software testing has been viewed as an optional activity, often performed late in a project with limited planning and executed carelessly (Mohan & Shrimali, 2017). As argued elsewhere, the techniques of Inayat et al. (2015) and Subramanian et al. (2017) use the software testing strategies to recover requirements from insufficiently documented software applications. De Souza et al. (2015) used requirements to generate test cases. All three approaches could be implemented together to test existing, older software systems to ensure their reliability under currently anticipated endeavors.

To this end, software testing can take up to 50% of software development time and cost; however, research has marginalized the importance of testing (Afzal et al., 2016; Beppe et al., 2018). Sanchez et al. (2018) acknowledged the deficiency with their own previously advanced method of generating performance regression tests on performance intensive software systems and suggested an improvement. Interestingly enough, effective software performance testing is fundamental to the development and delivery of quality software. The details of how the study was conducted appeared in the next section.

Transition and Summary

This section contained an introduction to the problem of software defects found at the end of the testing phase by software end-users. The purpose of this qualitative multiple case study was to explore the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. Lehman's laws of software evolution as the underlying conceptual framework provided a unique lens by which to view testing strategies used and the perceived benefits for adopting the strategy within government contracting organizations in the United States.

Section 1 commences with the foundation of the study and a discussion on the background of the problem. This section presented the problem statement, purpose statement, the nature of the study, the research question, the interview questions, and the conceptual framework. Moreover, Section 1 further elaborated to include the definition of terms, assumptions, limitations, delimitations, the significance of the study, contribution to IT practice, and the implications for social change. In the end, Section 1 concludes with the literature review, which provided a discussion on existing literature and explored research applicable to software testing, testing strategies, and software evolution.

Section 2 commences with a reminder of the purpose statement to provide the reader in a logical, explicit manner, an understanding of the research. Section 2 continues with a further discussion on the role of the researcher, participants, the research method and design, population sampling, and ethical research. Moreover, Section 2 explores data collection instruments, data collection techniques, and data analysis. In the end, Section 2 concludes with a discussion of reliability and validity in the context of the study. Section 3 contains an overview of the study, presentation of findings, application to professional practice, implications for social change, recommendations for action, and recommendations for future research.

Section 2: The Project

This section begins with a reminder of the purpose statement, followed by a discussion that acknowledges my role as the researcher and provides an overview of the participants involved in the study. Then, I provide detailed information about the research method and design, followed by discussions on population and sampling, ethical research, data collection instruments, data collection techniques and data organization techniques, and data analysis. In the end, Section 2 concludes with a discussion of reliability and validity in the context of the study and transitions to Section 3.

Purpose Statement

The purpose of this qualitative multiple case study was to explore the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. The target population consisted of software developers from three government contracting industry organizations located along the East Coast region of the United States. I performed the data collection process by interviewing software developers that have experience using and supporting software testing strategies. The contributions of this study may help foster a greater understanding on the part of software developers to improve testing strategies to ensure the reliability of software applications in the government contracting industry. Thus, the research findings might contribute to positive social change by possibly improving the everyday life of citizens because of the improvement in the reliability of software applications in the government contracting industry.

Role of the Researcher

This section of the study describes the role of the researcher. The researcher’s role when conducting qualitative research is to collect quality data. Yin (2018) reported that for qualitative research designs, the researcher is the primary collection instrument. As the sole researcher, I was the primary data collection instrument. This role allowed me to design the study, develop insightful interview questions, collect data in the form of naturalistic reports, confirm participants' responses and ensure the understanding between the research and the participant eliminating personal bias from the study.

According to Berger (2015), an understanding between the researcher and the study area makes the research more holistic. Furthermore, Fusch and Ness (2015) are right that one of the challenges researchers often face during data collection and analysis is to mitigate bias. My professional experience in the information technology field for more than 15 years, and my experience as a software tester in the government contracting industry since 2002 gave me a holistic perspective of the study. Mitigating bias in research is a challenge. Sohn et al. (2017) reported that bracketing is a technique by which researchers set aside their knowledge, beliefs, values, and experiences to understand participants’ experiences. I used bracketing during interviews to ensure that I do not incorporate any personal bias into the research study.

Ethical behavior toward potential participants is necessary while conducting research. I performed ethical research and data collection analysis for this study. The Belmont Report, published by the National Commission for the Protection of Human Subjects of Biomedical and Behavioral Research in 1978, provided guidelines for the ethical treatment of participants. To ensure the study met ethical requirements, I followed the directions of the Belmont Report. Hammer (2016) acknowledged that the three fundamental principles for studying the protection of human subjects include (a) respect, (b) beneficence, and (c) justice. These principles were achieved through informed written consent, assessing risks and benefits, and the selection of participants (U.S. Department of Health & Human Services, 1979). According to Forster and Borasky (2018), the regulations of the Belmont Report reported that the principle of respect for persons is revealed in the informed consent requirement. To ensure that the study met moral requirements, as the researcher, I received a certificate for completing the Protecting Human Research Participants online training course issued by the National Institutes of Health (NIH; certification number: 171957) to protect and ensure the privacy of all participants (Appendix A).

Trust builds rapport. Building rapport is vital to any interview (Lucas et al., 2017). Leins, Fisher, Pludwinski, Rivard, and Robertson (2014) observed that interviewing may shed light on the process and the way experts deal with critical incidents. My role as the researcher allowed me to build rapport with participants. The answers provided shed light on the process and the way participants' deal with critical incidents. According to Leins et al. (2014), the interview protocol is an instrument of inquiry where the researcher asks specific questions about a topic, gaining reflection and truthful answers from study participants. I used an interview protocol as an instrument of inquiry for asking specific questions about a topic, obtaining reflection, and truthful answers from study participants (Appendix B).

Participants

This section of the study describes the eligibility criteria for participants in this study. The eligibility requirement was an important factor when considering potential research participants. Yin (2018) explained that potential research participants should be knowledgeable of the topic and able to provide suitable answers to the research questions. For this study, the eligibility requirements included (a) must have at least 2 years software development experience, (b) must be currently employed by a government contracting organization located along the geographical East Coast region of the United States, (c) must have software testing experience or knowledge, and (d) must not have a recurring working relationship with me. Software developers in the government contracting industry who met these criteria should have the capability to answer questions and provide clarity.

The Institutional Review Board (IRB) is a board designed to approve, monitor, and review behavioral research involving humans. Dukes et al. (2015) explained that each step of the IRB process is in place to ensure the scientific quality of the study and the ethical conduct of the research team and the research participants. I obtained approval from Walden University’s Institutional Review Board (IRB) before contacting my potential participants. My study’s IRB approval number was 03-19-19-0583689. Researchers may rely on a professional network to gain access to study participants (Borgers, Pownall, & Raes, 2016). I reached out to my professional network, using LinkedIn to locate potential government contracting organizations. Peticca-Harris, de Gama, and Elias (2016) identified mediators as employers or managers of the organization who would help the researcher to gain access to eligible participants. Along the same lines, Fischer-Lokou, Gueguen, Lamy, Martin, and Bullock (2014) noted that a mediator could increase the trust between the researcher and the participants because of their relationship with colleagues at the participating organization. I contacted the mediator, introduced myself, and explained the purpose of the study, and asked for a signed letter of cooperation. Cacari-Stone, Wallerstein, and Minkler (2014) agreed that participants are more likely to agree to participate in a study if the research question is relevant to their field of study and may result in helping their organization policy wise.

After receiving the letter of cooperation from the mediator as part of the IRB approval process from Walden University, the mediator helped me to identify the participants who met the eligibility requirements and then sent their information to me via a separate email. According to Grieb, Eder, Smith, Calhoun, and Tandon (2015), the relationship between researcher and study participant should be defined. Building trust and positive rapport are vital. Some researchers found that building trust establishes a working relationship by keeping participant information private (Hoyland, Hollund, & Olsen, 2015; Nakash, Nager, & Maymon, 2015). Moreover, Drabble, Trocki, Salcedo, Walker, and Korcha (2015) urged us to establish a working relationship with participants; the participant needs to know more information about the researcher, the study, and the allotted interview time. Once receiving the participants’ information from the mediator, I extended an invitation to eligible participants an informational email, including the consent form, which explained the purpose of the study, the procedures, any risk, and benefits for participating in the study and the confidentiality of participants. I asked all research participants to read and reply to the consent form to ensure anonymity and confidentiality of all participants in compliance with Walden University's IRB requirements. Once participants responded electronically to the consent form, I began scheduling interviews. Each participant had an opportunity to ask questions via email, or via telephone before the start of the interviews to ensure that they are comfortable with the interview process. According to Haahr, Norlyk, and Hall (2014), researcher and participant interaction during the interview process influences trust and confidentiality. When I scheduled interviews with participants, I summarized the interview process to ensure comfort with the process. I reminded participants that their participation in the study is 100% voluntary and that their participation and organization name would remain confidential throughout the study.

Research Method and Design

Method

This section of the study elaborates more on the discussion of the research method and identifies the specific research method used in the study. The 3 types of research methodologies include qualitative, quantitative, and mixed methods. Every kind of methodology has its advantages and disadvantages. Hence, the design that a researcher uses is based according to preference.

I chose a qualitative research method to explore and understand the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications. According to Njie and Asimiran (2014), qualitative research involves the studied use and collection of empirical studies. For this qualitative research, I reviewed relevant empirical studies that related to my study. Qualitative research answers questions about the ‘what,’ ‘how’ or ‘why’ of a phenomenon rather than ‘how many’ or ‘how much’ (McCusker & Gunaydin, 2015). I chose the qualitative research method because I wanted to explore and understand ‘how’ the testing strategies used by software developers in the government contracting industry ensured the reliability of software applications. Qualitative research was relevant for exploratory studies, and it stimulated further research on a larger scale (Cronin, 2014). Stake (1995) supported the qualitative research method adding that it is valid for qualitative case studies as being holistic, empirical, interpretative, and emphatic to understanding a phenomenon. For this study, I chose a qualitative research method because it allowed for a thorough understanding of the testing strategies used by software developers in the government contracting industry to ensure the reliability of software applications.

There are two other methodologies that I could have selected for my study; they are quantitative and mixed methods. Quantitative research places focus on the ability to test a hypothesis using statistical data (Barnham, 2016). I did not select a quantitative research method for this study because my focus is not on the ability to test a hypothesis using statistical data. According to McCusker and Gunaydin (2015), quantitative research addresses a phenomenon using statistical numerical data and mathematical methods. For this study, statistical numerical data would not address the intended focus of the research question. Thus, I did not use statistical numerical data to explain or discuss the phenomenon. In contrast, Khan (2014) argued that the quantitative method is primarily associated with research in the natural sciences. I did not use the quantitative method to explore the natural sciences since my goal was to explore the holistic phenomena and real-life experiences of participants.

I considered using mixed methods research for this study. Mixed-methods research involves the combined use of qualitative and quantitative data in a single study (Halcomb & Hickman, 2015). According to Charman, Petersen, Piper, Liedeman, and Legg (2015), a mixed-methods approach may be used when neither a quantitative nor a qualitative method supports the comprehension of the study. While the quantification of data was not required to support the understanding of this qualitative research, neither quantitative nor the mixed methods approach was appropriate for this study. In contrast, McCusker and Gunaydin (2015) reported that the collection of data increases as a result of the intense combination of methods. As a result, I did not select the mixed methods approach because of time consumption when combining methods. Overall, the qualitative method is appropriate for this research because it addressed the intended focus of the research question.

Research Design

This section of the study elaborates more on the discussion of the research design and identifies the specific research design used in the study. The case study research design was chosen for this study. According to Yazan (2015), case study research was one of the most frequently used qualitative research methodologies. I selected a case study design for this qualitative research study because it is the most commonly used in qualitative research. The traditional design of qualitative research includes case study, ethnography, phenomenology, and narrative (Percy, Kostere, & Kostere, 2015). I chose a case study because it is a common design to explore and understand the central research question. Dasgupta (2015) claimed that case study research is useful when a phenomenon is broad, complex, and cannot be studied outside the context in which it occurs. When conducting a case study, Yin (2018) reported that it might be realistic to collect data from at least two of the following six sources of evidence: documentation, archival records, interviews, direct observations, participant observation, and physical artifacts. The reason I chose a case study design for my research because the information from interviews and documentation as sources of evidence provided an understanding of a broad research topic.

The ethnography research design was considered for this study. Some researchers use ethnography designs to demonstrate how cultures react, social implications, or communication between groups or other individuals (Ross, Rogers, & Duff, 2016; Trnka, 2017). I did not select an ethnographic design for this study because the demonstration of how cultures react, social implications, or the communication between groups or other individuals was not the intended focus of the study. According to Keutel, Michalik, and Richter (2014), an ethnographic research design is the preferred method of choice when the objective is to understand a culture. I did not select an ethnographic design because my research question does not require the study of culture.

The phenomenological research design was considered for this study. A phenomenological research design acquires lived experiences and events from the phenomenon (Blackmon, 2017). I did not select a phenomenological design for this study as understanding lived events from the phenomenon is not the intended focus of the research questions. According to Kruth (2015), phenomenological research is the investigation of human experiences through the eyes of people that are living the phenomenon. For accurate results, the interviewer should have a minimum of 20 participants when considering a phenomenological design (Canli & Demirtaş, 2018). I did not select a phenomenological design for this study because understanding the lived experiences of individuals is not the intended focus of the research question.

The narrative research design was considered for this study. A narrative research design involves storylines from participants that address sequences of events, specific activities, and causes and effects (Leedy & Ormrod, 2015). I did not select a narrative research design to involve the storylines from participants. When researchers like Wolgemuth (2014) and De Loo et al. (2015) explored research designs, they indicated that narrative research designs study the lives of individuals and provide stories about their lives. I did not select a narrative design for this study as understanding the lives of individuals is not the intended focus of the research question. Thus, a narrative research design was not an appropriate fit for this study. As I reflected on the probable designs, the multiple case study design was suitable for this research because it addressed the intended focus of the research question.

Population and Sampling

This section of the study describes the population and discusses the sampling method chosen and identifies how data saturation was achieved. Moreover, this section discussed the setting for the semistructured interviews. The population for the study included software developers working for government contracting organizations located along the East Coast region of the United States. Berger (2015) claimed that the population characteristics in a qualitative research study relate to participants' subjective experiences with the phenomenon. For this study, software developers were selected for the population as they have the experience and the knowledge necessary to answer the central research question. According to Robinson (2014), the first step in the data collection process is to identify the study population by using inclusive and exclusive criteria. The population included software developers who had software testing experience or knowledge and worked for a government contracting organization.

Purposive sampling is used in qualitative research when researchers explore the perspective on a specific research topic. Beverly, Hamel-Lambert, Jensen, Meeks, and Rubin (2018) reminded us that total population sampling is a type of purposive sampling, where the entire population is included in the research because they meet the criteria. Moreover, Daniel (2014) insisted that researchers implement purposive sampling in qualitative case studies so that researchers can explore the participants' perspective on a specific topic. I chose to use total population purposive sampling for this study to explore the participants' perspective on the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. According to Dasan, Gohil, Cornelius, and Taylor (2015), total population purposive sampling is used to validate measures of commonality. I chose a total population purposive sampling for this study to validate measures of commonality.

In contrast, Robinson (2014) reported that qualitative research is not appropriate for random sampling as a statistical sample of the universe is the intended focus of the research question. I did not select random sampling for my study because a statistical sample of the universe is not the intended focus of the research question. However, Ojo and Popoola (2015) acknowledged that the total population purposive sampling is considered appropriate based on the fact that the population size is relatively small and shares the same characteristics. For this study, I adopted a total population purposive sampling technique to capture as broad a spectrum of experience as possible. Thus, the total population of software developers representing three different government contracting organizations was 10.

This section of the study identified how data saturation was achieved in the study. According to Morse (2015), data saturation in qualitative research occurs when no new or relevant information can be captured with additional interviews. Gentles, Charles, Ploeg, and McKibbon (2015) explained that the number of participants could vary widely depending upon the depth of information obtained from well-crafted interview questions. In this study, data saturation was achieved through the collection of multiple sources of data, which included interviews and organizational documents that focus on the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. Fusch and Ness (2015) noted that there is no number of participants that would guarantee data saturation. Meanwhile, Malterud, Siersma, and Guassora (2016) suggested that researchers should continue interviewing qualified participants until data saturation is reached. I kept interviewing qualified participants until no new information emerged.

Ethical Research

This section discusses measures to assure that the ethical protection of participants was adequate, the informed consent process, procedures for withdrawing from the study, and incentives for participating in the study. The design of this qualitative multiple case study may reduce potential ethical risks. Thus, to assure the ethical treatment of research participants and the appropriate conduct of investigators, Walden University required that all researchers seek the approval of the Institutional Review Board (IRB) before the collection of data commences. I obtained the approval from Walden University's IRB before data collection at the participating government contracting organizations. My study's IRB approval number was 03-19-19-0583689. I abided by the three primary principles of the Belmont Report, which included: respect for persons, beneficence, and justice. According to Forster and Borasky (2018), the principle of respect for persons stresses that researchers consider an individual's right to determine whether to participate. I adhered to the requirements of the Belmont Report and considered an individual's right to decide whether to participate in the study. The process of obtaining informed consent for research participation, as reported by Biros (2018), is one method that attempts to secure the ethical rights of potential research applicants. Above all, providing consent forms allows participants the opportunity to understand all aspects of the study before deciding to participate (Schrems, 2014). I invited potential participants a chance to participate in the study via email. The email contained a copy of the informed consent form which explained the purpose of the study. According to Chiumento, Khan, Rahman, and Frith (2015), the informed consent process is designed to ensure the rights of all participants are not violated in any way. Beskow, Check, and Ammarell (2014) supported sending invitations and assuring confidentiality to participants. I emailed the study invitation and noted the assurance of confidentiality to participants. Gelinas, Wertheimeir, and Miller (2016) reported that the primary function of consent in human research is to protect and advance the interests of potential research participants. I presented each participant with a consent form and explained that participation is voluntary, and participants have the option to withdraw from the study by notifying me at any time.

This section discussed the procedures for withdrawing and incentives for participating in the study. Each participant involved in the study is informed that participation is entirely voluntary, and they have the right to withdraw from the study at any time without consequences (Gibbins, Bhatia, Forbes, & Reid, 2014). Beskow et al. (2014) believed that there is no obligation for participants to continue participating in the study should they feel uncomfortable. I reiterated to participants before the start of the interview that participation in the study is 100% voluntary, and they may withdraw from the study at any time without any consequences. According to Yip, Han, and Sng (2016), any incentives for participating in the study should be made clear to the participants before the start of the study as part of the consent process. Although recruitment is the overall process of selecting suitable participants for a project, Robinson (2014) reminded us that when recruiting participants for an interview study, the decision to offer a financial incentive in exchange for a participant's participation should be taken into consideration by the researcher. Furthermore, the disadvantage of offering an incentive in exchange for participant participation may fabricate or falsify their interview responses to gain a monetary award. Therefore, I reiterated to the participants involved in this study that there would be no incentives offered for participation.

This section discussed and explained how the names of individuals or organizations are kept confidential. As the researcher, I protected the confidentiality of all participants. Again, to ensure ethical research practices, researchers must protect the confidentiality of all participants (Singhal & Bhola, 2017). I explained clearly the goals of the interview and the expectations of the participants before initiating interviews with each participant. The National Commission for the Protection of Human Subjects of Biomedical and Behavioral Research (1978), as reported in the Belmont Report, indicated that the data which links the information to the participants who provided the information should be protected then destroyed at the earliest convenience when it is no longer needed. In the consent form, I explained how I safeguarded the data obtained during the study by keeping it in a locked safe at my financial institution, which I stored under lock and key for 5 years. Only I would have access to the locked box and possession of the key. Saunders, Kitzinger, and Kitzinger (2015) suggested the use of pseudonyms to create a degree of anonymity to protect certain areas such as participants' names, religion, cultural background, place of residence, occupation, and any other identifiable characteristics for the participant. I assigned a pseudonym for each participant in protecting their identity within the study. I also assigned a pseudonym for the participating organizations, and I did not identify the name within the study. I maintained the names of participants and organizations in a password protected Universal Serial Bus (USB) flash drive that is separate from the actual study data. The actual study data is stored on a second password protected USB storage device. Both devices are stored at my financial institution and destroyed after five years.

Data Collection

Instruments

This section of the study identifies the primary data collection instrument, explains how the technique was used, and discussed how reliability and validity enhanced the data collection instrument process. In this qualitative case study, I was the primary instrument for data collection, as explained in the Role of the Researcher. Per Yates and Leggett (2016), researchers are the primary data collection instruments in qualitative studies, and they must work directly with the data. According to Ridder (2017), data collection is based on triangulation, where interviews, documents, and observations are combined. I triangulated the data for this study using interviews and a combination of documents. The documents that I used for this study included organizational documents to identify the types of testing strategies used. Ridder (2017) also explained that qualitative data could be collected in the form of in depth semistructured interviews to gain a holistic understanding of the research question. When Beskow et al. (2014) conducted their research, they found that an interview with open-ended questions helps with the avoidance of unresponsiveness during the interview process and minimizes bias. As the primary data collector and qualitative researcher, I used an interview protocol (see

Appendix B) to collect data using open-ended questions for this study. Per Patel, Shah, and Shallcross (2015), interview protocols are instructions interviewers follow to ensure the consistency between interviews that increases the reliability of the study. The interviews enabled me an opportunity with each participant to ask specific follow up questions that may contribute to the collection of rich data. Yin (2018) reported that documentation could be used to expand further and confirm the data collected from interviews. I used organizational documentation provided to expand further and confirm the data collected from the interviews. The organizational documentation would consist of meeting notes or minutes, test plans, test cases, test logs, and test summary reports. There was no pilot study for this research.

Member checking is a technique for ensuring credibility. Conducting member checking during each interview ensures the reliability and validity of the data collection process (Marshall & Rossman, 2016). I followed the interview protocol to promote a positive professional relationship with each participant and to extract relevant information that was essential to answering the central research question. Caliz, Samaniego, and Caliz (2016) noted that the purpose of member checking is to allow each participant an opportunity to confirm or deny the interpretation of the data. For this study, I asked follow-up interview questions to each participant of the study allowing the opportunity to confirm or deny the interpretation of the data.

Data Collection Technique

This section of the study discusses the technique used to collect data, described the advantages and disadvantages of the data collection process, and identified how member checking was used for this qualitative study. Interview data for this study was collected using an interview protocol (see Appendix B). According to Wood, Burke, Byrne, Enache, and Morrison (2016), the interview protocol guides the researcher in the direction for conducting professional interviews. O'Cathain et al. (2014) noted that interviews are the recommended approach when working with professionals. I collected recorded open-ended interview information from both telephone or Skype professionally using an interview protocol. The use of the phone for conducting interviews is becoming more popular in data collection. Carter, Bryant-Lukosius, DiCenso, Blythe, and Neville (2014) found that the use of the telephone as an alternative to face-to-face interviews reduces personal beliefs and biases. Zhang, Woud, Velten, Margraf, and Kuchinke (2017) identified the benefits of phone interviews as part of data collection, including a low refusal rate, convenience, and low cost. I conducted telephone interviews because of participants' location for convenience, low refusal rate, and low cost. According to Sipes, Roberts, and Mullan (2019), Skype can be an effective method for collecting detailed information from participants. I conducted interviews using Skype to collect detailed information from participants when the telephone was not feasible.

As for any data collection technique, there are advantages and disadvantages the researcher must consider. In most cases, an advantage to using a semistructured interview for qualitative research is the need to use attentive listening and probe for clarity while conducting semistructured open-ended interviews to ease the interview process (Gibbins et al., 2014). Nevertheless, another benefit of using a semistructured interview as a data collection technique is that they concentrate more on the case study topic (Bowden & Galindo-Gonzalez, 2015). Finally, as one final advantage, a researcher gains by conducting semistructured interviews is the opportunity to observe nonverbal communication (Seitz, 2016). The interviews for this study were semistructured using open-ended interview questions, and I observed nonverbal communication (see Appendix D).

In contrast, as for any data collection technique, there are disadvantages to using a semistructured interview. In 2014, Baskarada noted that researcher bias and the misrepresentation of data collected during the interview process could potentially taint the results of the study. Hence it is critical to ensure that no bias affects the data collection process. I ensured that there was no bias involved so that it does not taint the results of the study. The interviews for this study were semistructured using open-ended interview questions.

Member checking is a technique used to enhance the reliability and validity of the data collection instrument. According to Goodell, Stage, and Cooke (2016), member checking techniques ask participants to review the findings to enhance the reliability and validity of the data findings. Following each interview, the recorded files were transcribed and annotated with a summary sent via email to participants for review and verification, followed by member checking activities. Morse (2015) suggested that member checking is a crucial step in establishing validity and reliability in a qualitative study. Moreover, Marshall and Rossman (2016) indicated that conducting member checking during each interview ensures the reliability and validity of the data collection process. For this study, the use of member checking helped enhance the reliability and validity of the study.

Data Organization Techniques

This section of the study describes the technique used to organize the data and to discuss how to store it securely. The organization of data in a qualitative case study requires the use of specific practices due to the amount of information and evidence collected during the study. Yin (2018) pointed out to ensure the validity and reliability of a study and to expose themes and patterns; a researcher uses research notes, research logs, and interview transcriptions. Moreover, other researchers suggested the use of reflexive journaling to summarize and track the experiences encountered (Cuellar, 2018; David & Hitchcock, 2018). For this study, I practiced the technique of reflexive journaling to document thoughts and perceptions before and during the research process.

Research logs are effective tools for recording information. According to Merriam (2014), qualitative researchers use research logs to make a note of obstacles encountered along with ideas emanating from data collected. Meanwhile, Yin (2018) observed that researchers use notes to document preliminary data interpretations. I used a research log to document emerging themes and patterns and trends from the data. As noted by Lakshmi (2014), researchers use research logs to (a) minimize potential bias, (b) provide a valuable audit trail for conformability, and (c) identify challenges that might occur during the study. I used research logs to reduce the risk of potential bias and as an audit trail. Plus, I labeled and categorized the research log entries based on notes from the interview and company documents. Dennis and Walcott (2014) reported that the organization of data is an essential process if a researcher expects to have a meaningful study. For a meaningful study, I organized the data collected according to patterns and themes. The identification of citations and references used in this study are stored using the citation manager Mendeley, and the use of the NVivo software application to store, file, and organize the collected research data.

Data collected from the interviews were transcribed and coded in a Microsoft Word file. For each completed interview file, I organized the results from the telephone interview into themes to promote research results promptly. Thus, to conceal the identities of participants of the study, all interview and audio files received the naming convention Organization 1 - Participant 1, Organization 2 - Participant 1, and so forth. According to Brennan and Bakken (2015), thematic analysis is one of the most known forms of data analysis in qualitative research.

Further, Brennan and Bakken (2015) emphasized NVivo as the data management tool to identify emerging themes from narrative passages. Furthermore, the transcribed data for each interview audio recording was stored in a password protected folder using the same naming convention. For these reasons, researchers must maintain good data organization practices to protect their participants. Hashem et al. (2015) explained that confidential information should be stored in a secure location for five years upon completion of the study and then disposed of as soon as possible. I saved all data on a password-protected USB flash drive at my financial institution for five years. Afterward, I would purge all forms of data, including password-protected USB flash drives, field notes, interview audio transcriptions, and collected documents relevant to the study.

Data Analysis Technique

This section of the study identified the appropriate data analysis process for the research design and discussed the specific data analysis technique used for this study. As noted by Derobertmasure and Robertson (2014), researchers need to identify and analyze their data to interpret the research findings correctly. Lawlor, Tilling, and Smith (2016) explained that the rigor of qualitative research helps to establish the trustworthiness of the data and includes the use of well-established data collection and analysis techniques, including the use of triangulation. Moreover, Patton (as cited in Yin, 2018), reported that the four types of data triangulation used to validate the findings of a case study include data, investigator, theory, and methodological triangulation. I selected methodological triangulation for this study.

Researchers use data triangulation to increase the validity of inference in qualitative and quantitative research. Data triangulation, as described by Scheibe et al. (2018), involves the use of different types of people or groups to get multiple perspectives of the data, whereas, theory triangulation consists of the use of various theories to analyze data (Fusch, Fusch, & Ness, 2018). I did not select data triangulation as a data analysis technique for this study because the analysis of multiple perspectives from different groups of people is not the intended form of triangulation that would support this study. Plus, I did not select theory triangulation because the analysis of multiple theories is not the intended form of triangulation that would support this study. According to Carter et al. (2014), investigator triangulation involves using multiple researchers in the same study to provide different perspectives on the same data. I did not select investigator triangulation because, as the sole researcher, I do not have access to additional researchers who might support exploring the phenomenon of this study. Finally, methodological triangulation involves using multiple sources of data found within one design (Zhao & Chen, 2018). I selected methodological triangulation for this study because I used multiple sources of data for this case study design. Yin (2018) reminded us that the use of multiple sources of data, a researcher can triangulate the data more accurately. I used multiple sources of data collection to provide a complete understanding of the phenomenon. I collected data from multiple data sources to gain as much data as I could regarding the phenomenon of testing strategies for this study. According to Carter et al. (2014), method triangulation is frequently used in qualitative studies and may include interviews and other methods of data collection, all regarding the same topic. Also, Ryan (2013) reported that examining company documents can enhance the quality of interviews. For this study, I began by conducting individual interviews, followed by document analysis as methods of data collection. I reached out to the mediator to obtain additional information that helped identify the organizational documents that were beneficial to my study. The documents of interest included meeting minutes, test cases, test plans, and any other documents related to my study.

Data analysis focuses on uncovering key concepts from raw data. According to Clarke and Braun (2018), thematic analysis involves a three-step coding process: preparation, organizing, and reporting. The initial step of the data analysis process is preparation. This step includes the reviewing of each interview and member checking transcript to gain a holistic understanding of the raw data. Morrison and Luttenegger

(2015) stressed the importance of triangulating interview data with additional sources. Plus, Morse (2015) explained how data triangulation becomes increasingly important to enhance validity and reduce bias in data collection for qualitative research. Furthermore, Marshall and Rossman (2016) suggested three sources of triangulation, including (a) open-ended semistructured interviews, (b) direct observation of data collection, and (c) company documents. Direct observation of data collection uses other data collection procedures, such as surveys and questionnaires; however, they have the least effect. For this study, the sources of triangulation that I used included open-ended semistructured interviews and company documents as provided by the mediator. Furthermore, researchers gain a greater understanding of their study through the observation of data from different perspectives (Salmona, Kaczynsk, & Smith, 2015). I reviewed relevant and available information posted on the company's website, capturing research notes that were used later in the generation of codes.

The next step of the data analysis process is the organization of themes. Research data need to be organized to identify the most efficient and effective methods of observation. As part of the data analysis process, researchers need to carefully examine and inspect the quality of their interview data (Langham et al., 2016; Yin, 2018). As the researcher, I carefully examined and inspected the interview data of each participant. Some researchers noted to ensure that participants correctly answer a research question, the results need to be organized by themes (Low, Crawford, Manias, & Williams, 2016; Ranney et al., 2015; Sutton & Austin, 2015). According to Neuman (2014), codes are brief symbols that represent the crucial topics present in the data and are developed by the researcher to expose such areas as events, relationships, situations, and opinions. Following each interview, I played back the audio recordings on a digital voice recorder, then manually transcribed and coded it into Microsoft Word. Coenen et al. (2016) noted that compiling the data into a central resource would provide a holistic view of commonality across the data. In the same manner, Vaismoradi, Jones, Turunen, and Snelgrove (2016), found that the selection of themes during the data analysis process is a fundamental task of the researcher because commonly used words or phrases by participants might link the theme to the research question and conceptual framework. Thus, the organization of data through categorizations of participants, interview questions, or additional documented sources validated the analysis from various perspectives. I organized the data and then coded into themes.

Upon completion, I re-examined the themes with participants and made any necessary adjustments according to feedback received by participants. Once the data is confirmed accurate, researchers use qualitative data analysis software programs (QDAS) to support their research. NVivo is one of several software packages from the QDAS category (Estrada & Koolen, 2018). Often, researchers identify the benefits of NVivo as supportive of data management and its capabilities to code and organize themes (Maher, Hadfield, Hutchings, & de Eyto, 2018). I used Nvivo version 12 software to complete all interview data and documentation collected to save time for material organization and theme identification. As reported by Kaefer, Roper, and Sinha (2015), the use of NVivo demonstrates how software tools can promote analytical flexibility by improving the transparency and trustworthiness of the qualitative research process. After the identification of emergent themes, I shared the results with the participants as a member checking technique. Member checking, as suggested by Stillwell et al. (2018), provides participants with the opportunity to comment and provide feedback or express disagreement with my interpretation of their responses. I ensured that the themes aligned with the conceptual framework of Lehman's laws of software evolution and the literature review on testing strategies by analyzing the results of the coding using NVivo version 12 before generating a report of my findings.

Finally, the last phase of the data analysis process involved the generation of a report. The report illustrates theme patterns that trace back to the literature review and the conceptual framework. The NVivo software was a crucial component during the data analysis phase as it provided help in the generation of the final report. The presentation of findings is explored further in section three of the study.

Reliability and Validity

The following section of the study introduces reliability and validity and identifies similar criteria for qualitative research. Lincoln and Guba (1985) proposed four commonly used criteria for ensuring rigor in research (a) dependability, (b) credibility, (c) transferability, and (d) confirmability. In research, both reliability and validity are techniques used to ensure transparency of a study and to minimize bias (Singh, 2014). Reliability and validity have numerous meanings in qualitative and quantitative research

Reliability

The term reliability in qualitative research refers to how one addresses dependability. I used the interview protocol listed in Appendix B and the triangulation of methodological evidence to ensure reliability. A study is reliable and dependable, according to DeGirolamo, Di Pillo, Porto, Todisco, and Barca (2018) when the results are repeatable, and the verification of data references are accurate. I used reflexive journals and research logs during the study to document reproducible results and to verify the accuracy of data references. Henningsen, Sort, Møller, and Herling (2018) confirmed that the research log is used as an audit tool, allowing the researcher to identify and reflect on challenges that may occur during the research study. I used a research log as an auditing tool to help track any obstacles encountered during the study.

Dependability

The term dependability in qualitative research refers to consistency. For this reason, researchers enhance the dependability of the study through the process of member checking, transcript review, expert validation of the interview questions, or interview protocols to confirm and validate the data findings (De Massis & Kotlar, 2014). In the same manner, Crowe, Inder, and Porter (2015) reported that researchers could establish dependability by providing records of an audit trail of the methods and procedures for the study. For this study, the research is dependable, and the reader will comprehend how the researcher was able to derive the findings of the data.

I used a research log and reflexive journal to record my process so that any reader or reviewer can understand the decision-making process for each situation encountered. Thomas (2017) proposed that researchers use member checking methods as an approach to deviate the incorrect data findings and to assure dependability. Daniel (2018) suggested the technique of member checking to ensure that the researcher's interpretation of the data is dependable and accurately captures the participant's perspective on the phenomenon. Through the lens of dependability, I incorporated member checking as a method to validate the research findings.

Validity

The term validity in qualitative research refers to the credibility, transferability, and confirmability of the data findings. Marshall and Rossman (2016) reminded us that the concept of validity asks the question: does this research process measure what it claims to measure? Validity in research is the extent to which an instrument is measured. There are two types of validity in research: internal and external (Bartels, Hastie, & Urminsky, 2018). Internal validity refers to the causal claims in the setting where inferences regarding cause-effect or causal relationships (Reeves et al., 2018). In contrast, external validity refers to data findings that apply to more extensive population settings or groups (Glasgow, Huebschmann, & Brownson, 2018). Hence, for this qualitative study, validity helped reach data saturation to assure the credibility, transferability, and confirmability of the data findings.

Credibility

The term credibility in qualitative research refers to acknowledging the truth. Credibility involves ensuring that the data findings from the study are credible from the perspective of the participants in the research (Stockman, 2015). For this multiple case study, I used member checking to confirm my interpretation of each interview. Credibility establishes trustworthiness and rigor. According to Lincoln and Guba (1985), member checking is the most crucial technique for creating credibility by allowing the participants to verify the accuracy and credibility of the researcher’s account of their experiences. Caretta (2016) suggested member checking to confirm that the researcher’s interpretation represents the intent of the participants’ comments. Moreover, when used correctly, member checking can increase the trustworthiness of the study while adding value such as credibility to the research (Becher & Wieling, 2015). In this study, I used the member checking technique to confirm the accuracy and credibility of the participants' experiences during data collection.

Transferability

The term transferability in qualitative research refers to the relevancy of the data findings to other settings. The transferability of a study is evident when the outcome uses a different context or group (Rapport, Clement, Doel, & Hutchings, 2015). In 2015, Barnes noted that transferability is achieved when the data findings of the study have meaning to individuals not involved in the study. In contrast to external validity, transferability does not include broad claims (O'Sullivan & Conway, 2016). Lub (2015) reported that the data findings could be theoretically transferable to other contexts if researchers provide rich detail with a complete description of the case study. Thus, through the lens of transferability, I provided complete descriptions for future readers to determine whether they can apply these practices to future research studies.

Confirmability

The term confirmability in qualitative research ensures that the data results can be confirmed and supported by others. Arundell, Mannix, Sheehan, and Peters (2018) agreed that confirmability occurs when the researcher confirms that the participant's views represent the data without any bias from the researcher. Further, Tong and Dew (2016) explained that the data findings and interpretations reflect the opinions of the participants. I used the defined interview protocol during the interview to ensure the participant's responses are confirmable, and bias is minimal. I used the data findings and interpretations to reflect the views of the participants. Moon, Brewer, Januchowski-Hartley, and Blackman (2016) concluded that researchers must demonstrate that the data findings connected to the data can be replicated as a process. I documented the data findings from interviews and the steps taken during each phase of the research process in a reflexive journal to provide repeatable steps for succeeding reviewers. Qualitative researchers can enhance the confirmability of a study by conducting follow-up member checking and asking questions from numerous perspectives (Singh, 2014). For this study, each participant was given an opportunity during follow-up member checking to confirm or dispute the interpretation of his or her responses.

Data Saturation

The term data saturation in qualitative research occurred when the data findings collected produce no additional information. Fusch and Ness (2015) wrote that researchers could ask multiple participants the same questions as one method to reach data saturation. Also, Gibbins et al. (2014) wrote that when qualitative researchers receive no additional information after conducting several interviews with research participants that researchers achieved data saturation. To ensure data saturation, I interviewed each research participant until no additional information replicated the phenomenon of the study. Qualitative researchers can achieve data saturation through the lens of methodological triangulation using multiple sources of data and member checking methods to verify the accuracy of the interview data (Cope, 2014; Fusch & Ness, 2015). Roy, Zvonkovic, Goldberg, Sharp, and Larossa (2015) insisted that the quality or the depth of the data reflects saturation. I used methodological triangulation and member checking to ensure data saturation.

Transition and Summary

The purpose of this qualitative multiple case study was to explore the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. In Section 1, I discussed the background of the study, the problem statement, the purpose statement, and the nature of the study. I included in this section, my assumption as a researcher, the research limitations, and delimitations. The section continued with the research question and a discussion on the conceptual framework. The academic literature concluded this section.

In Section 2, the purpose of this qualitative multiple case study was restated for providing the reader with a broad perspective of the nature of the project. I began Section 2 with a discussion regarding the role of the researcher, participants, and research method and design, which was then followed up with a discussion on population and sampling strategy used to select participants. Next, the ethical responsibilities that are required by the IRB were explained, followed by another discussion of the data collection and analysis techniques, along with data organization techniques and instruments chosen. The collection of data from phone interviews and Skype, along with organizational documents, were explained even more. I used the qualitative data analysis computer software package NVivo to organize and analyze my data. Methodological triangulation was used to ensure data saturation. The section concluded with a discussion on reliability and validity in the context of the study. In Section 3, I present the findings from my research, describe applications for professional practice, address implications for social change, make recommendations for future research, and offer reflections.

Section 3: Application to Professional Practice and Implications for Change

This section contains information from the qualitative multiple case study, including a presentation of the findings from the data collection and a description of how this study may be significant to IT practice and society. Then, I discuss information from my study that encourages positive implications for social change. Finally, I conclude Section 3 with suggestions for future work as well as personal reflections related to the study.

Overview of Study

The purpose of this qualitative multiple case study was to explore the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. The data for this study were collected from two primary sources: semistructured phone interviews and organizational documentation.

Ten participants were recruited through total population purposive sampling that included software developers from three respective government contracting organizations located along the east coast region of the United States. Moreover, I collected and analyzed 77 organizational documents for use in the study to illustrate the work performed within the organizations. The participants ranged in status from junior to senior level software developers. All of the participants of the study had between 2-35 years of software development experience and knowledge of software testing practices. Most had between 5 and 35 years of software development experience. Two participants had less than 5 years of software development experience.

I categorized participants into two groups by software development experience, with eight participants having between 5 and 35 years of experience and two participants having less than 5 years of experience. I also categorized participants into two groups by software testing experience with seven having between 5 and 35 years of software testing experience and two participants having less than 5 years of experience. The remaining participant did not discuss their software testing experience. I organized themes by major and subthemes associated with a significant theme. Additionally, reference counts are based on attributions to theme keywords.

Similarly, a reference may be specific to one theme or include two or more themes in the same reference. My analysis of the data resulted in four major themes to emerge during the data analysis phase of this qualitative multiple case study: (a) communication and collaboration with all stakeholders, (b) development of well-defined requirements, (c) focus on thorough documentation, (d) focus on automation testing. The findings from this study are comparable to the findings revealed in the literature review. Furthermore, the findings from this study support the use of Lehman's laws of software evolution as the conceptual framework. In the following section, the four major themes revealed during the data analysis phase was explored and synthesized for the reader.

Presentation of the Findings

When I first started this qualitative multiple case study, I wanted to answer the overarching research question: What testing strategies do software developers use to ensure the reliability of software applications in the government contracting industry? In this section of the study, I present and introduce four major themes that emerged during the data analysis phase of the study. I conducted semistructured phone interviews, which were member checked to ensure transcription accuracy and also to enhance the methodological triangulation process. Furthermore, methodological triangulation was used to analyze the two sources of data obtained, which included semistructured phone interviews and organizational documentation.

The interview and member checking activities, along with the organizational documents were all analyzed and uploaded into the qualitative data analysis software tool NVivo, where the analysis of four major themes emerged from the study. The identification of these four major themes provided potential strategies that could be used for implementing testing strategies in government contracting organizations. According to Bonello and Meehan (2019), qualitative analysis software tools such as NVivo provide the researcher with an audit trail to visually analyze and code the data through various iterations, annotations, as well as mapping concepts into themes. The development of themes during the data analysis phase was identified, and the findings were tied back to the existing literature review and conceptual framework.

In the following section, the four major themes that emerged during the data analysis phase are compared to the existing literature review, and the findings are then tied back to Lehman's laws of software evolution, which served as the conceptual framework for this study.

Theme 1: Communication and Collaboration with All Stakeholders

The theme communication and collaboration with all stakeholders was the first theme to emerge during the data analysis phase of the study. The theme emerged based on the responses of all participants, an analysis of organizational documents, and confirmed by previous and current research. Within this theme, several subthemes were mentioned by the participants, in the organizational documents, and identified in previous research that contributed to communication and collaboration with all stakeholders. Based on the participant interviews, communication and collaboration with all stakeholders are critical in the culture of software testing as it lays the groundwork for software testing. Supporting the participants' views was the study by Allison and Joo (2015) reported in academic and professional literature. Allison and Joo indicated that laying the groundwork for software testing through communication and collaboration with all stakeholders requires working together as a team to understand the requirements for testing the software application. Also, supporting the theme was a study by Berman and Chutka (2016), cited in the academic and professional literature of this study. Berman and Chuka reported that communication is not just restricted to talking, but also to listening and nonverbal communication. Kim, Seo, and David (2015) showed that communication connects people by face-to-face or written communication, collaboration with all stakeholders ensures that everyone has an opportunity to provide input into analyzing the problem and formulating an effective solution. The researchers' views were consistent with the opinions of the participants, who indicated that the team should strive to have as many face-to-face meetings as possible. While it is critical to have accurate communication when testing, collaborating with all stakeholders such as supervisors, system analysts, other software developers, software testers, and end-users is crucial to avoid producing a defective software application.

The responses from all 10 participants indicated the importance of laying the groundwork of software testing through communication and collaboration with all stakeholders. Seven participants reported that the attendance at daily stand-up meetings to exchange ideas through communication is beneficial and an excellent way to disseminate software testing information. The remaining three participants reported that while their organization does not currently have a test team, they rely on peer code reviews to examine and evaluate the content and quality of the software application. While peer code reviews promote the development of working software through communication and collaboration with all stakeholders, peer code reviews have the most significant impact on code quality, coding style and standards, and testing (Sun, Wu, Rong, & Liu, 2019).

An analysis of 11 organizational documents supporting this theme included a charter document outlining the purpose, goals, roles, responsibilities of members, processes, task tracking, and meeting frequency. From the literature, Yague, Garbajosa, Diaz, and Gonzalez (2016) reported that communication is critical in the exchange of information between team members. Moreover, Strandberg, Enoui, Afzal, Sundmark, and Feldt (2019) reported to make informed decisions, practitioners need information from software testing before the execution of test cases and continue even after the software testing phase is complete. According to Wang, Graziotin, Kriso, and Wagner (2019), software testing requires communication between all team members. Organization 3-Participant 2 stated, “I think communication between all teams from the project manager to the business analyst, to the development team to the stakeholders and the software experts, the communication should always remain open." In a separate study, Alzoubi, Gill, and Al-Ani (2016) reported that poor communication is a significant risk to the project when testing software. The fact that delivering incomplete, inaccurate, or inadequate messages could cause severe software problems leading to unreliable software applications and delayed software delivery. A case in point, when asked about the current project and to explain the testing process, Organization 3-Participant 3 stated,

> before creating version2, the process was everywhere and unorganized. People had their hands on the documents, and there was no communication. Nothing. What I did was made sure that the approval process was seamless and user-friendly, making sure that it went to the proper people, proper steps, and proper phases. So now, when I test, I usually have some test users, and some users are heavily involved in the actual 'live' ones such that whenever they have time, I have them make a 'dummy' process, and then I typically test it myself.

The findings of this study demonstrated that communication and collaboration with all stakeholders are in alignment with existing literature. According to Rola, Kuchta, and Kopczyk (2016), increasing communication among all stakeholders of a project leads to a more reliable identification of performed tasks. According to Organization 8-Participant 3, increasing communication with all stakeholders is essential. From the literature, Bellery, Hodges, Camp, and Aduddell (2016) found that communication is essential to teamwork. As explained earlier, communication and collaboration with all stakeholders lay the groundwork for software testing. Organization 8-Participant 1 stated, "we have daily stand-up meetings every morning; this helps us to understand what other people are working on, and it also informs the product owners and the clients." Support for this idea also exists in the literature, as Wohlin et al. (2015) noted that the communication between teams improved, and so did the collaboration with the stakeholders. Organization 8-Participant 2 stated, "we have a testing team, and as tasks are completed, what we as developers do is unit testing." Research carried out by Jan et al. (2016) showed that software testing has taken on the interest of developers, testers, and end-users.

Communication and collaboration are the skills used to help teams build stronger relationships and understand their work better. For example, one participant explained that communication and collaboration with all stakeholders are both relevant to software testing stating, "it helps the team to collaborate and manage their work better." Kropp, Meier, and Biddle (2016) argued that experience leads to collaboration showing that successful teams tend to use more collaboration practices when testing. Organization 3-Participant 4 stated, "if a tester does not understand what they are testing and you cannot explain it to them, and they are not getting what it is, at some point managers step in and say what the answer is and they proceed from there." In the culture of software testing, both communication and collaboration are essential. Weidner, Pauwels, McGuire, and Davis (2017) claimed that communication and collaboration help bring projects up to speed more quickly while passing along insightful tips. The insightful tips might require using basic code testing, unit testing, regression testing, or user acceptance testing as the software development team develops and delivers more software applications in a short time, through communication and collaboration with all stakeholders, the testing team could verify and validate them using integration testing or regression testing. Organization 3-Participant 4 talked about the various tips in the tote bag that developers use. Organization 8-Participant 1 explained,

> as a developer, we do perform some form of unit testing. It is basically the code testing before we pass the software to the testing team. We also make sure that the new code does not break the previous code that was built. Then, we also do some form of regression testing. As for user-acceptance (UA) testing, we usually do that manually before sending it to the testing team.

Recent literature further supports the theme communication and collaboration with all stakeholders as a strategic testing strategy that software developers could use to ensure the reliability of software applications in the government contracting industry. According to Organization 8-Participant 3, during the last Friday of the third week, planning meetings with stakeholders are arranged to discuss every phase of the project. Crevier and Parrott (2019) confirmed that communication and collaboration with and between stakeholders should be encouraged through all phases of the testing project. As noted by Ramanathan, Faulkner, Berry, et al. (2018), when thinking about future roles, more communication and collaboration with all stakeholders could help achieve the end goal. Their study found that an increase in communication and collaboration with all stakeholders led to new visions and testing ideas. In the end, effectively leveraging efforts demonstrates how an organization makes full use of the testing resources currently available at their disposal.

The theme communication and collaboration with all stakeholders align with Lehman's laws of software evolution, which served as the conceptual framework for this study. One of the characteristics of Lehman's laws of software evolution is the law of complexity. According to Lehman (1996), as software evolves, its complexity increases unless work is done to maintain or reduce it. As noted earlier, communication and collaboration are two components designed to work together because they both have laid the groundwork for software testing. For projects that are complex in nature, communication and collaboration are necessary. Lehman et al. (1997) demonstrated in their empirical studies that communication and collaboration rely on continuous improvement. Hence, collaboration is unachievable unless communication begins. Mashia, van Wyk, and Leech (2019) agreed that communication is a priority. One participant summarized the overarching theme stating,

> I am going to say this again. Communication is the number 1 priority. Number two, build a relationship between the project manager and ensure that he or she understands that project. Also, building a relationship with the business analysts and developers is critical. Everyone is working as a team. I have worked with developers in the past where they would stop what they are doing to ensure that QA has a complete understanding of the code designs. It is important to the project and the company.

The data in Table 2 lists the subthemes of communication and collaboration with all stakeholders. The study participants' identified these subthemes as results of their experiences encountered in various projects at each of their respective organizations.

Also, Table 2 highlights the number of participants and the number of references in the organizational documents supporting the subthemes.

Table 2

Subthemes of Communication and Collaboration with All Stakeholders

	Participant		Document	
Major Theme	Count	References	Count	References
Communication and collaboration with all stakeholders	10	100	11	160
Effectively Leverage Collaboration	10	35	7	171
Create Project Transparency	9	33	4	149
Obtain Essential Feedback	9	21	4	96

Effectively leverage collaboration. Collaboration requires communication. Also, collaboration brings groups together to focus their efforts on achieving a common goal. When viewed through the lens of software testing, effective collaboration shapes the way teams work together in pursuit of a common goal. The responses from all 10 participants indicated that effectively leveraging collaboration improves testing efficiency and brings the team closer together. Their views were consistent with the findings of Dadkhah, Araban, and Paydar (2020). The viewpoints of Organization 3-Participant 2 and Organization 4-Participant 3 on effectively leveraging collaboration enabled team members to generate more productive and innovative ideas for software testing a product.

An analysis of seven out of the 11 organizational documents supported the subtheme of effectively leveraging collaboration defining workflow processes, task tracking, and tools used, such as Jira, to deliver product owner objectives. Support for these ideas exists in academic and professional literature, as Strandberg et al. (2019) noted that the flow of information in software testing is related to communication. The authors noted that effectively leveraging collaboration seems to help locate issues faster. From the literature, Tissenbaum (2020) reported that making and accepting suggestions are important aspects of collaboration, providing opportunities for participants to relate their understanding to the problem. According to Organization 3-Participant 4, their group has daily scrum meetings where the entire team is at those meetings to discuss issues encountered with the project. Research carried out by Kitamura, Alegroth, and Ramler (2017) reported that the goal of collaboration is to transfer knowledge, exchange experiences, and enrich the understanding of the opportunities and challenges between the two sides.

The findings of this study demonstrated that effective leveraging collaboration is in alignment with existing literature. Anderson-Cook, Lu, and Parker (2019) agreed that having an effective collaborating team could accelerate the problem-solving process. Communication is essential for effective collaboration and keeps testing projects on schedule and stakeholders in the loop. Effectively leveraging collaboration is one means of ensuring communication and collaboration with all stakeholders. Wang et al. (2020) reported that effective communication might be the foundation required for collaboration. Previous researchers agreed that collaboration was linked to timely communication, which allowed team members to stay updated on the progress of the project while making contributions to achieve common goals (Collette et al., 2017; Luetsch & Rowett, 2016).

Previous research carried out by Bell, Murray, and Davies (2019) confirmed the findings for this theme. Bell et al. provide insight into fostering collaboration and showed evidence that might be used to tailor future projects. The findings from these researchers were consistent with the responses from Organization 3-Participants 1, 2, 3, 4, and Organization 8-Participants 1, 2, and 3 of the study. These participants indicated that the Agile software development collaboration technique used within their organization encouraged everyone to work as a team. To that end, Douglas-Smith, Iwanaga, Croke, and Jakeman (2020) indicated that collaboration amongst users is encouraged, and users can contribute to the project.

The conceptual framework that guided this study, Lehman's laws of software evolution, supported the findings of this study. Lehman (1996) reported that the functionality of software increases overtime to maintain user satisfaction. Contributions by Kour and Singh (2016), which was cited in the professional and academic literature of this study, also supported the findings. Kour and Singh reported that easy, flexible, and earlier testing, along with quick deliveries, increases cooperative collaboration, communication, and coordination by delivering high-quality products to the customer. Organization 8-Participants 1, 2, and 3 indicated that through positive communication and effective collaboration, projects are finished sooner and the ability to promote high-quality products. Research carried out by Ferrell and Ferrell (2016) showed that high-quality products are achieved through positive collaboration.

Create project transparency. Project transparency is required to understand what might be wrong while testing a troubled project. When viewed through the lens of software testing, like communication and collaboration with all stakeholders, project transparency is imperative. Six of the participants' responses indicated that through the use of tools like Jira, combined with daily stand-up face-to-face meetings, and involvement in activities like these create project transparency and reduces the risk of producing poor testing results. Their views were consistent with the findings of Pauly, Michalik, and Basten (2015). Pauly et al. reported that daily stand-up meetings make teams more productive and effective at testing. The meetings allow team members an opportunity to discuss and inspect the progress of their work and remove any obstacles encountered while testing. Moreover, Pauly et al. (2015) explained that each team member answers the following three questions during the daily stand-up meeting: What have you worked on since the last meeting? What will you work on until the next meeting? Have you encountered any obstacles? When team members acknowledge these questions, they can address the concerns encountered during testing. Furthermore, the daily meetings serve as a reminder and foster the creation of transparency of the work in progress, as well as communication and collaboration with all stakeholders involved in the testing phase.

An analysis of four out of the 11 organizational documents supported the subtheme of creating project transparency. This information was consistent with the views of the six participants discussed earlier. The organizational documents provided identified strategies used for creating project transparency that was discussed during daily meetings. Support for these ideas exists in academic and professional literature, as Sanchez-Morcilio and Quiles-Torres (2017) noted that better communication and the creation of project transparency are shown in daily meetings. The authors noted that as a result, there is more cooperation and support among team members. Although user involvement is relatively high, the daily meetings were allowing the possibility of re-planning and adjusting the project as opposed to traditional project management techniques.

Jira is an issue tracking tool used to document defects while testing. The tool is visible to all stakeholders and enhances efficiency. Organization 8-Participants 1, 2, 3 reported that Jira is used to monitor the workflow and to track the progress of a particular item. When a project is substantial enough to require an official project turnaround rather than deciding to fix the software defect, it is critical to recognize project transparency during the testing phase. According to Kaur and Kaur (2019), Jira is a test management and quality metrics tracking tool designed with the functionality to create, plan, and execute tests. Also, Jira allows team members to distribute tasks across their team. Even more, the tool can prioritize and track the team's work in full context with complete visibility. Organization 8-Participant 1 stated, "… we record everything that we do into Jira." Liu, Eisingerich, Auh, Merlo, and Chun (2015) noted that transparency matters because organizations find it difficult to hide information when things go wrong. In their research, Liu et al. (2015) performance-tested transparency, and the results showed that transparency has positive effects when testing a project. For example, Organization 4-Participant 1 talked about having a second set of peer's eyes on the software project is helpful.

In today's organizations, numerous testing activities introduce project transparency. Nine of the participants talked about creating project transparency through the lens of software testing in their interviews. One participant pointed out the use of Visual Studio in their SharePoint environment to create and manage most of the tests performed to ensure that every request made was completed. Konnola et al. (2016) highlighted the importance of transparency, communication, and collaboration, allowing the opportunity for team members to obtain a better understanding of their work and the work of other team members. Glaser and Strauss (1967) developed a theory titled the grounded theory. The purpose of the grounded theory is to inductively generate theory that is grounded in or emerges from the data. The theory identified three conventional methods used in grounded theory: participant observation, interviewing, and collection of artifacts and texts. Though creating project transparency emerged from the data through interviews, a detailed understanding of the grounded theory is required to make that conclusion.

Obtain essential feedback. Obtaining essential feedback is also a critical component of the testing process, as reported in Table 2. Soliciting feedback from everyone involved in testing the project would help improve the quality and aid in the deployment of a reliable software application. Nearly all of the participants' responses indicated the importance of obtaining essential feedback in their interviews. On this point, three participants reported that the time spent on testing was a significant source of feedback. The remaining six participants indicated that feedback is an essential element of testing practices. From the literature, Strandberg et al. (2019) reported that fast feedback is the right approach for improving the flow of information and communication in software testing. Although software testing produces non-trivial information to manage and communicate, the flow of information in software testing involves numerous feedback loops. The authors also proposed using automation frameworks and testing in a simulated environment as strategies for obtaining faster feedback and dealing with test results efficiently and effectively. Previous research carried out by Xiao et al. (2018) showed that feedback offered when software testing is valuable. As Organization 3-Participant 2 explained, “we have a weekly team meeting to discuss any issues going on, and projects that are being worked on within the scope of testing and coordinated with management. So, it is our meeting. A time where we can hash out problems with the methodology process, testing, data, and things like that and so that the manager can get involved to keep both development and testing moving forward.” In another study, Heeager and Rose (2015) showed that providing feedback at the beginning of the testing phase meets business needs.

An analysis of three out of the 11 organizational documents supported the subtheme of obtaining essential feedback by identifying a plan for implementing improvements that would be beneficial to future work, which was supported by existing literature (Nidagundi & Novickis, 2016). When viewed through the lens of software testing, feedback is crucial to understanding how to add value to a project. Previous research by Beller (2018) corroborated the findings for the study. Beller reported that software developers yearn for feedback, be it from their peers, code reviews, or local execution of their tests. When asked about how much time is allocated for testing, one participant explained that if their team is given the average amount of time for testing and it does not seem correct, some form of feedback would be issued. With feedback at the center of today’s software development practices, the flow of information in software testing is built on numerous feedback loops (Strandberg et al., 2019).

The findings of this study demonstrated that obtaining essential feedback is in alignment with existing literature. Scatalon, Barbosa, and Garcia (2017) stated that feedback based on testing code coverage is useful. Essential feedback and effective communication are necessary for the team when testing code. Through essential feedback, information is provided continuously on tasks to guide the testing process. In the culture of software testing, Zhou et al. (2018) found that test cases are centered on feedback information collected during the testing process. Moreover, previous research carried out by Zhou et al. showed that feedback information obtained on early fault detections is beneficial for further improving the cost-effectiveness of testing.

When viewed through the unique lens of Lehman's laws of software evolution, feedback is the eighth law. Lehman (1996) applied feedback from users to solicit ideas for new enhancements. One participant reported that when designing code, following the requirements are essential feedback to meeting the client's needs. Another participant reported as feedback that the accurate measure of how effective the testing strategy used is based upon the bug release rate. According to Panichella and Molina (2017), test effectiveness metrics show a percentage value of the difference between the number of bugs found and the overall number of bugs found in the software. In the literature, Inayat et al. (2015), and Prechelt, Schmeisky, and Zieris (2016) all explained the importance of obtaining feedback as early as possible. Organization 8-Participant 1 stated, "right now, we are using an agile methodology. It is a methodology that we use based on the feedback of the product owner." According to Godfrey and German (2014), Lehman recognized that the processes involved in developing and maintaining software appeared to form a feedback system, where the environment provided a signal that had an intense impact upon the continued evolution of the system. To that end, Dhandapani (2016) explained that while software testing provides the gaps in the committed functionality of the product, user testing provides input, which can provide the first round of feedback provided by the customer before the product is officially released.

Theme 2: Development of Well-Defined Requirements

The theme development of well-defined requirements was the second theme to emerge during the data analysis phase of the study. The theme emerged based on the responses of nearly all participants, an analysis of organizational documents, and confirmed by previous and current research. Within this theme, several subthemes were mentioned by the participants, in the organizational documents, and identified in previous research that contributed to the development of well-defined requirements. Software testing is more than a bug hunting activity, but an activity that determines if the criteria meet the required results. Based on the participant interviews, the quality of functional software requirements is crucial because it is nearly impossible to produce a high-quality implementation from a poor-quality design. A recent study found that poor representation of the software design can bring on numerous testing obstacles (Strandberg et al., 2019). The researcher's findings revealed that the lifetime of a test case from its creation to its retirement might influence how test results are communicated. One participant pointed out that test cases are derived from the requirements with an intent to reveal software defects. Organization 3-Participant 4 echoed a similar response and added that the goal is to have minimal defects. Petunova and Berzisa (2017) reported that these are efforts that reliable software applications must begin with before the start of application development.

Nine participants' responses indicated the development of well-defined requirements as an essential testing strategy before the start of application development in their interviews. One participant conveyed the importance of writing test cases based on the decomposition of the requirements before the start of application development. Two participants also conveyed that writing the test script before developing the code meets the best practices for requirements. The six remaining participants indicated that testable requirements make it possible to develop test cases before the start of application development to determine whether the condition meets the requirements.

An analysis of 26 organizational documents supported the theme of developing well-defined requirements (Table 3). From the literature, Antinyan and Staron (2017) concluded that a well-defined software requirements document is the backbone for high-quality software design. According to Organization 3-Participant 2, “a full and clear understanding of how the software works are imperative, adding that testing the software application is based upon the requirements.” Black-box testing is a testing strategy in which the data used for testing derive from the software requirements (Jan et al., 2016). A user story helps with the creation of a simplified description of a requirement. Organization 8-Participant 2 stated, “when the end-user wrote their story, the first step is to ensure that it meets the requirement.” Previous research by Lucassen, Dalpiaz, E. M. van der Werf, and Brinkkemper (2015) found that user stories are a widely used notation for formulating requirements. Researchers Lucassen et al. (2015) and Hooda and Chhillar

(2015) recognized acceptance testing as a testing strategy to confirm the acceptance of user stories. The goal for testing user stories is to confirm the acceptance of user stories from a holistic approach through the lens of customer requirements.

The findings of this study demonstrated that the development of well-defined requirements is in alignment with existing literature. As emphasized by Fortineau, Paviot, and Lamouri (2019), the business rules must ensure that the product meets all of the customer requirements. In a separate study, Fernandez et al. (2017) showed that not having well-defined requirements is the leading cause of requirements failure. Furthermore, Alahyari, Gorschek, and Svensson (2019) stated that not well-defined requirements might expose other factors such as an inefficient architectural design, which in turn may have long term side effects.

Recent literature further supports the theme development of well-defined requirements as a strategic testing strategy that software developers could use to ensure the reliability of software applications in the government contracting industry. Unterkalmsteiner, Gorschek, Feldt, and Klotins (2015) implied that involving requirements as a test strategy and test plan review would ensure the support of testing requirements correctly. Organization 3-Participant 2 talked about testing requirements and how the test planning was done very poorly in the past, observing that more bugs came out later than they should have. Similarly, Huang (2017) noted that the analysis of requirements and improving customer satisfaction are both challenging tasks. Consequently, requirements that are traceable and easy to comprehend so that stakeholders can develop a sense of knowledge are considered reasonable and without conflict.

The data in Table 3 lists the subthemes of the development of well-defined requirements. The study participants' identified these subthemes as results of their experiences encountered in various projects at each of their respective organizations. Also, Table 3 highlights the number of participants and the number of references in the organizational documents supporting the subthemes.

Table 3

Subthemes of Development of Well-Defined Requirements

	Participant		Document	
Major Theme	Count	References	Count	References
Development of well-defined requirements	9	25	26	173
Requirements are testable	9	19	22	22
Requirements are traceable	3	5	3	6
Requirements are clear	3	8	2	14

Requirements are testable. Well-defined requirements are testable, as reported in Table 3. Ideally, the central goal of testable requirements is to ensure that the quality is correctly maintained from the beginning. Also, the essence of testable requirements is to make it possible to develop tests to determine if the requirement has been met. Nine participants agreed that well-defined requirements should be testable. The responses from nearly all the participants indicated that requirements that are clear and concise are easily testable. From the literature, Inayat et al. (2015) reported that a well-defined requirements document is testable, traceable, and transparent defining everything the software must accomplish. Software testing is the execution of a software application against test cases (Lemos et al., 2018). Organization 3- Participant 2 stated, “When I do a project plan, I do my estimate based on how much testing has to be done based on the number of test cases and the number of resources that I have.” Lalitha, Latha, and Sumathi (2016) reported that test cases must be carried out before the start of the software design. Aceituna and Do (2019) argued that one of the toughest challenges for requirements is the lack of testing. Organization 3-Participant 1 pointed out, “we do not do a ton of that for obvious reasons, but that is what ensures that everything is 100% showing up on the label correctly.” Challenges for testing include a lack of time for testing as well as low availability of the test environment, consistent with Strandberg et al. (2019) study. These findings also supported the second theme of this study.

An analysis of 22 out of the 26 organizational documents supported the subtheme that requirements are traceable, providing test cases based on the decomposition of the requirements. With that in mind, as part of document analysis, a PowerPoint presentation was provided that presented an overview of the software development lifecycle. Huzoree and Ramdoo (2015) also supported the findings of this study. Huzoree and Ramdoo confirmed that there should be some finite cost-effective processes in place in which the requirements could be validated through testing. Also, Huzoree and Ramdoo recognized numerous challenges that prevent the successful development of software, including customer dissatisfaction, cost overruns, an increase in the cost of maintenance due to rework, and errors found in the software due to poor quality deliverables. Organization 3-Participant 4 stated, “if the goal is to have minimal defects, you want to spend time to figure out how to write the code and test it.” Furthermore, defective requirements can lead to defects found in the final software product, which is not desired by either the end-user or developer. Chen, Shang, Nagappan, Hassan, and Thomas (2017) found that fixing defects in later phases of software testing or after the delivery of the software can be challenging and costly. According to Organization 3-Participant 1,

> … we are not like your traditional organization because we are tied to a government contract. So, for the current project, for the most part, our software shop is not 100% typical. A lot of what we do is bug fixes and enhancements on legacy applications. The customer oftentimes thinks that the software is designed to do something when, in fact, it is not. Therefore, what they think is a bug; yet is an enhancement. They want the software changed and not fixed. Thus, we do a scope creep constantly, which is you submit a bug for Problem A, and as we gather to fix Problem A all of a sudden, Organization 3 will try to get us to fix problems B, C, and D as well.

Shirazi, Kazemipoor, and Tavakkoli-Moghaddam (2017) explained scope creep as adding features and functionality to the scope of the software without discussing the consequences of it and the impact it has on testing. Shirazi et al. discussed the leading causes of scope creep to include poor documentation, poor change control, poor information transformation, and external changes all could have negative impacts on testing. As a result of continuous testing, the required changes can be identified much earlier to avoid the consequences of scope creep.

The findings of this study demonstrated that requirements are testable is in alignment with existing literature. Andrews, Alhaddad, and Boukhris (2019) identified a process for testing requirements using regression testing. They created rules for testing requirements based on various failure scenarios. In a separate study, Andrews, Elakeili, and Alhaddad (2015) reported that when testing requirements, efficiency improved by 65%. Previous research carried out by Dou (2016) revealed that testable requirements are the foundation for any development project. In the culture of software testing, when considering requirements verification, regression testing is needed as a balance between cost and code coverage. Dou further explained that requirements should be traced from the stakeholder to the test objectives.

From the perspective of Lehman (1996), he realized the need for software systems to evolve as a result of the requirements to operate in or address a problem that is significant to real-world activities. The first characteristic, the law of continuous change, suggested that systems must continually be adapted else they become progressively less satisfactory. Although requirements seem to expand, the development of well-defined requirements provides a blueprint for future applications. When asked about testing requirements, one participant stated, “some of our apps have built-in test award numbers or contract numbers where we could do all the application functionality for those subpieces so that we can do a thorough regression test. It would be nice if every app had that enhancement, but since some are commercial off the shelf (COTS) products, we could never change those to do so, but for future applications, it would be nice to think about that sort of thing.”

Organization 8-Participant 4 stated, “we really don’t collect any metrics on testing. We do know when something is being categorized as a bug or feature enhancement through our tracking system." Similar to traceability, requirements should also be testable. Lehman (1996) believed that stakeholders would perceive an E-type system to have declining quality unless it is rigorously maintained and adapted to its changing operational environment. The second characteristic, the law of declining quality, identified why requirements are tested. According to Lehman, the central elements of focus, discipline, and rigorous effort must be sustained during the life of the software product to reduce the number of defects that are introduced. For this reason, test cases are written to help find problems detected in the requirements or design of the software. Through rigorous testing, a strict entry and exit criteria are followed along with all possible combinations of test cases and test data. One participant described the processes involving rigorous testing as going down every corridor and opening every door, ensuring that no door is left unopened. As Lehman (1996) noted, software systems must endure continuous change, or they will become less useful. Therefore, if the software does not adapt to the changing needs of the business stakeholders and end-users, satisfaction will decrease.

Requirements are traceable. Requirements traceability has shown to be an essential contribution to organizations that make proper use of traceability techniques. Three participants conveyed that requirements should be traceable when testing because they represent the needs of specific product designs. The responses from the participants indicated that the primary purpose of traceability meets the expectations of the requirements. Their views were consistent with the findings of Murtazina and Avdeenko (2019). The viewpoints of Organization 3-Participant 2 and Organization 4-Participant 2 on requirements are traceable, suggested that in the culture of software testing, traceable requirements saves time, reduces costs, and improves the overall quality of the software product. Rempel and Mader (2017) acknowledged that requirements traceability leads to increased development effort and documentation workload, which can be compensated by reduced costs and higher quality. In their research, Rempel and Mader found that projects that implemented requirements traceability techniques performed on average 24 percent faster and created 50 percent more accurate results than projects that performed without requirements traceability.

An analysis of three out of the 26 organizational documents supported the subtheme that requirements are traceable. A PowerPoint presentation that was included with the organizational documents highlighted vital artifacts to explain the use of a traceability matrix. Other documents supporting this idea included the organizational documents entitled "Requirements Management Plan" and "Test Cases and Roles," which stated that "once requirements are prioritized, they become the template input for the requirements traceability matrix." Mattman, Gramlich, and Kloberdanz (2015) reported that a real understanding of how to formulate requirements and how to document requirements does not exist. Mattman et al. also noted that functional testing provides measures for all relevant factors through the use of traceability matrices. Organization 3-Participant 2 complicates matters further when stating, "the ability to produce metrics based on the severity of the defects is important. If what we see in one month of testing, or sprints, we see a high level of defects in a particular area, that is something that will raise a red flag. Therefore, using things like traceability matrices to ensure the user stories and bugs line up properly with the test cases." From the literature, Rempel and Mader (2015) reported that traceability is an essential quality of software requirements; for this reason, it helps reduce software maintenance costs by educating the developer who needs to resolve the defect. Organization 4-Participant 2 described traceability as a percentage of bugs found as a measurement. Support for this idea also exists in the literature, as Mattman et al. reported that despite the criteria, each requirement should be traceable. According to Organization 3-Participant 2, traceability matrices are used to align requirements with the test cases.

The findings of this study demonstrated that requirements are traceable is in alignment with existing literature. Traceable requirements help with test case verification, enabling to keep traceability links between test cases and functional requirements (Roldan, Vegetti, Gonnet, Leone, & Marciszack, 2019). Traceability is a significant quality of software requirements. Most research showed that traceable requirements improve the quality of both the design and code of complex functions (Ali & Lai, 2016; Chandani & Gupta, 2018). Bagheri, Garcia, Sadeghi, Malek, and Medvidovic (2016) indicated that developers have difficulty understanding complex software units, such that the software has to be updated regularly, which in part adds more complexity to the software.

When viewed through the unique lens of Lehman's laws of software evolution, the law of complexity is the second law. Lehman's laws of complexity suggest that software will become progressively more complex over time unless explicit work is conducted to reduce complexity. One participant reported running regression tests to avoid the risk of breaking complex code that has already been tested. When viewed through software testing, Rempel and Mader (2017) explained that traceable requirements are a critical element of any rigorous software testing process.

Requirements are clear. Clear requirements are essential to software testing. When viewed through the lens of software testing, the more time spent writing clear requirements, the more likely it is that the end product will function as expected. Moreover, the time spent on writing substantial requirements and test specifications leads to considerable time and money saved during the testing phase of the project. Three participants talked about the significance of clear requirements when testing software in their interviews. The responses from the participants indicated that regardless of how clear the requirements are, they must also be accurate. Tsunoda et al. (2018) found that successful software projects are the result of clear software requirements. Organization 3-Participant 2 pointed out, “the simplicity of clear, well-defined requirements should be so easy to understand that anyone should be able to pick them up and take off running where the work was last left off.” Murphy and Wright (2018) noted that the establishment of clear software requirements, objectives, and goals, and a realistic schedule are the three critical components for project success. When asked about the requirements document, one participant mentioned that it should be clearly written and well documented.

In contrast, unclear requirements are among the many challenges that impact the accuracy of software designs (Britto, Mendes, & Borstler, 2015). Nikiforova and

Bicevska (2018) pointed out a long list of reasons that might cause a project to fail, including unclear software requirements. According to Organization 8-Participant 3,

> sometimes the requirements are unclear. As a developer, I understand it differently, whereas the testers understand it another way. In that case, I design to 1 plus 2 equals 4, and they say that no 1 plus 2 equals 9. It is important to let the client know that the requirements document is not clear, and we have to figure it out. And at that time, everyone will be on the same page. Once fixed, the rest is easy.

According to Joppen, Enzberg, Kuhn, and Dumitrescu (2019), 60% of errors occur because the implementation of requirements is not clear. One participant indicated that requirements that are not clear bring confusion. Liebel, Tichy, Knauss, Ljungkrantz, and Stielbauer (2018) acknowledged that it is increasingly important to establish communication. Moreover, the authors indicated that a lack of communication could lead to vague or unclear requirements.

An analysis of two of the 26 organizational documents confirmed the subtheme that requirements are clear. The information in these documents was consistent with the responses from Organization 3-Participant 2, Organization 3-Participant 4, and Organization 8-Participant 3, indicating that requirements are clear, complete, consistent, and unambiguous. Bronckers, Roc’h, and Smolders (2017) agreed that it is often desired to have clear requirements. Research carried out by Pereira, and de F. S. M. Russo (2018) explored obtaining clear requirements based on customer input. The results showed that implementing clear requirements promoted communication between software development and testing teams and stakeholders involved in the software project.

The findings of this study demonstrated that requirements are clear is in alignment with existing literature. Contributions by Luckmann (2015), which was cited in the professional and academic literature of this study also supported the findings. Luckmann reported that the Standish Group's Chaos Report acknowledged that clear requirements are key success factors in IT projects. Also, in the culture of software testing, Alsaqaf, Daneva, and Wieringa (2019) specified the need for clear requirements when implementing test criteria correctly. Organization 8-Participant 3 stated, "I test using the requirements, and if everything is okay, I push to test. When I test, if there are no defects, then I have tested correctly, and my understanding of the requirements was understood correctly."

The conceptual framework that guided this study, Lehman's laws of software evolution, did not support the findings of this study. Glaser and Strauss (1967) developed a theory titled the grounded theory. The purpose of the grounded theory is to inductively generate theory that is grounded in or emerges from the data. The theory identified three conventional methods used in grounded theory: participant observation, interviewing, and collection of artifacts and texts. Though ensuring that requirements are clear emerged from the data through interviews, a detailed understanding of the grounded theory is required to make that conclusion

Theme 3: Focus on Thorough Documentation

The theme focus on thorough documentation was the third theme to emerge during the data analysis phase of the study. The theme emerged from the responses of participants, an analysis from the organizational documents, and previous research. Within this theme, there were several subthemes mentioned by the participants, in the organizational documents, and recognized in previous research that contributed to the focus on thorough documentation. Based on participant interviews, policies and procedures, test plans and execution summaries, and maintenance logs are necessary to ensure that all processes flow correctly are tested and thoroughly documented. Thorough documentation is not software, but it is crucial to the software testing phase (Yadav & Yadav, 2015). The focus on thorough documentation can begin at the very start of the software process since the earlier the defect discovered, the less it would cost to fix. Moreover, thorough documentation makes testing easy and systematic. Nevertheless, poor documentation may affect the quality of the software or application, leading to poor testing results.

Eight participants talked about the need for thorough documentation to improve their testing process. One participant reported the need for thorough documents since these documents are publicly accessible and part of the Freedom Information Act. Three participants reported the need for thorough test plan documentation. The remaining four participants reported a general need for thorough documentation. From the literature, Aovak, Gugan, Varga, and Domotor (2018) explained that thorough documentation of principles is necessary for coordinating teamwork. Organization 3-Participant 1 stated, "developers, I should say a good developer will get the software application running and do what they possibly can to verify a defect." Support for these ideas exists in the literature as Yip et al. (2018), reported that thorough documentation ensures tasks are completed consistently, correctly, and are traceable.

An analysis of 13 organizational documents supported the theme of focusing on thorough documentation. Kramer, Brandt, and Borchers (2016) conveyed that thorough documentation is highly relevant for software testing tasks and increases software maintainability. Documentation testing is the testing of documents created before and after software testing (Itkonen, Mantyla, & Lassenius, 2016). In a separate study, Konnola et al. (2017) highlighted the importance of communication, collaboration, and transparency when testing and focusing on thorough documentation. Organization 3-Participant 4 stated, "because we work with testers, everybody comes to the program with different mindsets. If I get to the point where my basic tests are fine, then I hand it off to a tester, and they find problems, then we reassess my process to allow me to see that I missed something."

Organization 3-Participant 2 stated, "the problem with Organization 3 is that there are federal laws and rules that must be followed. So, they need much documentation." Another participant reported that their organization is documentation heavy. According to Organization 3-Participant 4, "there is very strict documentation, and it needs to be very thorough then approved by a manager. Once management reviews, make corrections, and approves the documentation, then it goes for technical review. After the documents have been reviewed and approved, they then become government documents." Patenaude, Pelletier, and Bingen (2015) noted that organizations should follow thorough documentation for policies and procedures in compliance with state and federal laws. Yoon, Dols, Hulscher, and Newberry (2016) confirmed that testing for accessibility could include functional testing, usability testing, and compliance testing. Previous researchers Iniesto, McAndrew, Minocha, and Coughlan (2016) recognized the importance of raising the awareness of accessibility challenges. The findings of their research confirmed that when focusing on thorough documentation through the lens of software testing, the testing process ensures that requirements are compliant with laws, policies, and regulations to avoid errors.

The data in Table 4 lists the subthemes of focus on thorough documentation. The study participants' identified these subthemes as results of their experiences encountered in various projects at each of their respective organizations. Furthermore, Table 4 highlights the number of participants and the number of references in the organizational documents supporting the subthemes.

Table 4

Subthemes of Focus on Thorough Documentation

	Participant		Document	
Major Theme	Count	References	Count	References
Focus on thorough documentation	8	29	13	333
Policies and Procedures	6	9	8	88
Test Plan and Execution Summary	2	3	4	73
Maintenance Logs	2	3	2	211

Policies and procedures. Policies and procedures are the decisive links between the vision of an organization and its daily operation. Researchers reported that thorough documentation is essential along with the mandate to follow procedures as documented and can be used to recreate works (Dyk & Meghzifene, 2017; Post, 2017). Six of the eight participants reported a general need for a thorough documentation of policies and procedures to optimize the software testing process. From the literature, Dyk and Meghzifene (2017) and Skoulis et al. (2015) found thorough documentation essential to policies and procedures. While test documentation improves communication about testing tasks and processes, thoroughly written documentation with a focus on policies and procedures helps teams to understand their level, scope, and types of testing strategies to use (Strandberg et al., 2019). One participant reported using the online documentation portal Confluence to create and share thoroughly documented policies and procedures related to the testing process in the event of a new hire. Craft (2019) sheds light on the use of online documentation portals to support the idea of thorough documentation for policies and procedures.

An analysis of eight out of the 13 organizational documents supported the subtheme of thorough documentation for policies and procedures. Eight organizational documents supported the idea of thorough documentation of policies and procedures. With that in mind, as part of document analysis, an organizational document titled “Determination/Reason(s) Statement” is specific to concise content information that explains the reasons behind the concluding decision. Chang, Seow, and Tam (2019) supported the findings of this study. They confirmed that policies and procedures are in place to reduce the risk of software errors. Also, they recognized that corrective policies and procedures address software risks to correct source code and remove bugs after bug detection. Organization 4-Participant 1 shared a similar view and added that nothing is ever full proof, thus having policies and procedures in place to prevent software errors from happening later on. Safa et al. (2019) supported the participant's statement acknowledging that policies and procedures are an effective and efficient method for preventing software errors from happening later on.

The findings of this study demonstrated that thorough documentation for policies and procedures is in alignment with existing literature. Budde et al. (2019) agreed that there is a need for thorough documentation for policies and procedures in order to be able to compare, relate, and replicate previous works. In a separate study, Schroder et al. (2019) indicated that keeping track of all information during testing is essential since thorough documentation is crucial. To that end, Goodman (2019) reported that thorough documentation makes it easy for novice users to get started.

The conceptual framework that guided this study, Lehman's second law of software evolution, supported the findings of this study. Lehman (1996) noted that the law of continuing change must be continually adapted, or it becomes less useful. When viewing thorough documentation for policies and procedures through the lens of Lehman's second law of software evolution, the differences between data derived from observations and computations may cause changes in the perception of the way software is implemented, its documentation, or both to change. Versteeg et al. (2016) reported that without the required knowledge, often writing thorough documentation for policies and procedures becomes challenging as a result of changes or incomplete information, causing the entire project to fail.

Test plan and execution summary. A good test plan helps organize and manage the testing effort. Therefore, obtaining stakeholder preferences during test planning is just as important as receiving feedback from stakeholders during the product design phase. When a test plan leaves out an essential requirement, it relinquishes a source of frustration for the next person who attempts to use it if not thoroughly documented (Hooda & Chhillar, 2015). One participant pointed out the importance of ensuring that test plans are thoroughly documented after observing more bugs come later on in the test phase than expected. From the literature, Wright et al. (2018) noted that as a best practice, the detail of the test plan should depend on the complexity of the designed logic. As defined by IEEE Standard 829, a section of the test plan is standardized to include the test plan identifier, introduction, test items, and features to be tested. Organization 3-Participant 3 stated, "the documentation workflow process was created to standardize the procedure in moving test plans and execution summary documentation through the whole approval process." In a separate study, Cazals and Dreyfus (2017) noted that thorough documentation consists of user and reference manuals. When viewed through the lens of software testing, user and reference manuals should be maintained regularly and thoroughly documented, providing support for efficient testing. One participant talked about providing a user guide with the delivery of a software application to give the user direction. The advantage of a user guide explains the rationale of each test and how to perform analysis (Nunes, Alvarenga, De Souza Sant'Ana, Santos, & Granato, 2015). Another participant reported that any test strategy used should always be tailored. Kukulies, Faulk, and Schmitt (2016) indicated that within test planning, test activities determine the test strategy. According to Afzal et al. (2016), the test strategy aims at a specific test method to meet the maximum test frequency.

An analysis of four out of the 13 organizational documents supported the subtheme of thorough documentation for test plans and execution summaries. As part of document analysis, an organizational document titled "Test Management Plan" provides a detailed and documented approach to validate and verify that each solution is delivered with confidence in the quality, integrity, scalability, reliability, and usability of the released implementation. Moreover, other organizational documents were supporting this idea to include an Execution Summary, which Ye, Zhang, Ruilin, Feng, and Tang (2019) indicated that the execution summary informs the user about the current executions of the test cases. One participant previously reported that the workflow process created to standardize the procedure in moving the test plans and execution summaries through the process for management approval. Kukreja, Singhal, and Bansal (2015) supported the findings of this study. Kukreja et al. emphasized that no software testing is carried out without a test management plan. The authors explained that a detailed test management plan provides answers about what to test by forming test cases for a project. Thus, software testing and the validation of software is a critical part of software quality.

Thorough documentation such as test plans and execution summaries has long been prominent on the list of best practices to improve software testing. Although thorough documentation may be described as an artifact intended to communicate information about the software product to end-users; however, the primary purpose is to produce reliable software, and that thorough documentation helps to achieve the goal.

From the perspective of Lehman (1996), he demonstrated that a program must continually adapt to the environment to maintain satisfactory performance. Lehman et al. (1997) demonstrated in their empirical studies that software that continuously adapts rely on thorough documentation. Skoulis et al. (2015) agreed that software that continuously adapts needs thorough documentation. When viewed through the lens of software testing, one participant reported, "in the development area; you may have developers making changes while testing is ongoing and that can cause a huge risk. Any changes made to code can affect all of the tests that have been completed, and this is why we thoroughly document and have regression testing because we do not know what was fixed."

Maintenance logs. Excellent record keeping is an integral component of thorough documentation. Through thorough documentation, maintenance logs can help understand past errors and provide clues that will prohibit future mistakes; otherwise, it would be difficult to understand (Hou et al., 2016). Two participants emphasized the importance of thorough documentation through the use of maintenance logs. From the literature, Gupta, Mehlawat, and Mahajan (2019) reported that maintenance logs measure accuracy and improve software testing. For example, Organization 3-Participant 4 stated, "the thing that I rely on the most is a running log of each of the steps in the program because if there are 500 steps in your process and its wrong, you want to know where it went wrong. So instead of guessing and you have the information for each of the 500 steps, you can look back to see where the problem is at." Arif-Uz-Zaman, Cholette, Ma, and Karim (2017) found maintenance logs to be immediately useful in improving the estimation of failure times for real-world assets.

An analysis of two out of the 13 organizational documents corroborated the subtheme of thorough documentation for maintenance logs. With that in mind, as part of document analysis, maintenance log files in the format of Excel spreadsheets captured the usage and maintenance of the system and workflow. Maintenance logs provide support during system maintenance while capturing anomaly and intrusion detection, as well as documenting software failure analysis. Shang, Nagappan, and Hassan (2015) indicated that maintenance logs could be used to understand code quality better. When viewed through the unique lens of Lehman's laws of software evolution, maintenance logs align well with the conceptual framework for this study. Lehman's law of complexity reflects the fact that with all maintenance, systems need to evolve due to its requirements to operate real-world activities (Lehman, 1996). There are two layers of effort that need to be addressed if the quality of the entire system is to be kept equal. The layers encompass code design and the integration of the coding work performed on the system in terms of code integration, documentation, adaptation of the design, and rework from other sections. Lehman explained that maintenance could refer to the upkeep effort that has to be expanded on the codebase.

The findings of this study demonstrate how the focus on thorough documentation improves software testing and aligns with existing literature. According to Post (2017), thorough documentation is necessary for understanding the past and also for recreating works. Organization 8-Participant 2 stated, “I think the biggest challenge is being able to recreate a problem. The hardest thing is when you have an issue, and you cannot reliably recreate it. That is how we tell the testing team. If you can reliably recreate it, then we will figure out how to fix it.” Support for this idea also exists in the literature, as Dosemagen, Liboiron, and Molly (2016) indicated that thorough documentation allows others to replicate experiments for more reproducible and transparent research.

Lehman’s laws of software evolution, which served as the conceptual framework for this study aligns with the findings of the theme focus on thorough documentation. The conservation of familiarity law and the law of complexity both support the theme of the study. Lehman (1996) found that during the evolution of software systems, the content of successive releases remained consistent because software developers needed to have a thorough understanding of the source code and behavior coining this as the conservation of familiarity law. In a separate study, Martin (2016) noted that a thorough understanding in many cases involves a detailed study of the phenomenon, which is accompanied by thorough documentation. On this point, Organization 3-Participant 4 stated,

> you have to go back to the step and find some information on what you are relying on and show it to the tester. It may be time-consuming, but it is important because you are helping the tester to become a better tester. Whether it is a challenge or not, you have to make certain that the tester has come up to speed. Also, as part of a developer’s job is to make sure that everyone is on the same page.

Equally important, Lehman's research showed that the law of complexity reflects the fact that with all maintenance, systems need to evolve due to its requirements to operate real-world activities. According to Organization 3-Participant 3, "maintenance is generally conducted once per month and performed after hours. I send an email to everyone indicating that the server will be going offline, and from there, my project manager then sends it out to everyone else to alert to save their work because the server will be going down for approximately 30 minutes – 1 hour depending on what the maintenance is." Gupta and Singh (2017) reported that software systems are becoming much more substantial and complex, containing several million lines of code and voluminous documentation, which makes comprehension of the system difficult. The existing literature reviewed for this study is in alignment with the findings of the study to make a case for focusing on thorough documentation.

Theme 4: Focus on Automation Testing

The theme focus on automation testing was the last theme to emerge during the data analysis phase of the study. The theme emerged based on the responses of all participants, the data analyzed from the organizational documents, and confirmed by previous and current research. Within this theme, there were several subthemes mentioned by the participants, in the organizational documents, and recognized in previous research that contributed to the focus on automation testing. Based on participant interviews, automation testing speeds up execution and reduces the effort of human involvement. Hanna, Aboutabl, and Mostafa (2018) confirmed that automation testing speeds up execution and could reduce the overall software testing time. Garousi and Pfahl (2016) noted that the goal is to identify undiscovered errors, not to prove that no errors exist. Garousi and Pfahl added that the automation of test activities is a popular approach in the software testing community. Meanwhile, Tramontana, Amalfitano, Amatucci, and Fasolino (2019) reported that automation testing might represent an effective solution to improve quality applications and to reduce testing costs. The researchers' views were consistent with the opinions of the participants, who indicated that one of the benefits of automation testing is improved code quality.

The responses of all 10 participants supported the idea of automation testing. According to Kononov and Rusakov (2018), automation testing helps to find bugs in software applications. Seven participants acknowledged that their organization is slow to automation testing. The remaining three conveyed that their organization's testing team supports automation testing. Tu, Lin, and Lee (2019) reported that automation provides data with higher quality and more efficiency.

An analysis of 27 organizational documents supported the theme of focus on automation testing by identifying a plan for implementing automation testing, which was supported by existing literature (Chandraprabha, Kumar, & Saxena, 2015). According to Meiliana, Septian, and Alianto (2018), automation testing is beneficial since it saves a lot of time and money. Meanwhile, Fadel et al. (2015) noted that one of the contributing factors to a slow transition to automation testing is due to the lack of skilled personnel. Organization 3 Participant 4 stated, "time is money and we get paid to write code and not to test." To that end, Meiliana et al. pointed out that the availability of tools or frameworks for automating tests is often not suitable for developer needs.

The findings of this study demonstrate how the focus on automation testing is in alignment with existing literature. Researchers Muller, Vette, and Horauf (2015) explained that adaptions are made to meet defined requirements. Then, Nouacer et al. (2016) showed that due to a lack of automation, software testing tends to consume 40%-50% of the development cost. According to Organization 3-Participant 4, "… the biggest drawback of testing is that it requires developers to have knowledge and experience and the know-how to do it." However, software testing researchers Brichni, Dupuy-Chessa, Gzara, Mandran, and Jeannet (2017) argued that the most significant factor should be to automate and quickly integrate into the considered environment. Organization 3-Participant 1 stated, "the honest answer is that we do not do much testing because of the environment that we are in." Although all participants from Organization 8 mentioned that automation testing is the responsibility of the testing team, one other participant in particular stated,

> I wish we did have automated testing because we can frequently run that in case of any changes outside of our app that we did not know were coming. We can have our test suite running, perhaps every day. So, I wish there were easier ways to implement automated testing and cheaper. I know that there are other suites out there, but apparently, we cannot afford them.

Supporting the participant's views was the study by Kumar and Mishra (2016), industry experts suggested that software tests executed only a few times are best left for manual execution, while those software tests that require large amounts of data and run more frequently are best automated. In summary, the existing literature reviewed for this study is in alignment with the findings of the study to make a case for a focus on automation testing.

Automation testing is a technique for improving software quality. In the culture of software testing, teams should focus on laying the groundwork for automation testing through communication and collaboration with all stakeholders. As discussed earlier, while it is critical to have accurate communication when testing, collaborating with all stakeholders such as supervisors, system analysts, other software developers, software testers, and end-users is crucial to avoid producing a defective software application (Wang et al., 2019). Then, develop well-defined requirements and determine which test cases to automate is essential since it is impossible to automate all levels of testing.

Lehman's laws of software evolution, which served as the conceptual framework for this study aligns with the findings of the theme focus on automation testing. According to Lehman (1996), as time proceeds, software needs to be evolved continuously to provide user satisfaction and to meet user requirements. Moreover, Lehman's laws of software evolution characterize the way that software applications will become progressive between maintenance activities and unit tests. Organization 8-Participant 2 stated, "primarily, I unit test. Pretty much just following the requirements to do a thorough unit test." On the other hand, Organization 3-Participant 3 stated, I do not use load testing, but for me, the use of unit testing and performance testing would be considered maintenance." Pawlak and Poniszewska-Maranda (2018) reminded us that unit tests verify the smallest independent and testable parts of the source code. When viewing through the lens of Lehman's law of software evolution, the idea focus on automation testing supports the law of complexity, which suggests that as software evolves, its complexity increases unless work is done to reduce it. Organization 3-Participant 4 stated, “we strive to do as much testing as possible; however, the current project is starting out manual, so now I am looking into making it more efficient using automation.” In the end, the conceptual framework Lehman’s laws of software evolution proved to be a helpful guide to retrofit the focus as a strategy to ensure the reliability of software applications in the government contracting industry.

The data in Table 5 lists the subthemes of focus on automation testing. The study participants’ identified these subthemes as results of their experiences encountered in various projects at each of their respective organizations. Furthermore, Table 5 highlights the number of participants and the number of references in the organizational documents supporting the subthemes.

Table 5

Subthemes of Focus on Automation Testing

	Participant		Document	
Major Theme	Count	References	Count	References
Focus on automation testing	10	37	27	98
Test Cases	10	34	26	84
Lack of Automation Testing	9	20	2	62
Unit Testing	7	13	5	73
Open Source Automation Tools	5	12	3	50

Test cases. Automation testing requires a test case for the requirement understudy to be verified. When viewed through the lens of software testing, test cases check regression issues against the latest code changes in order to improve the efficiency and quality of the software. The responses from all 10 participants indicated the critical role test cases play in the automation testing process. According to Em and Reedy (2015), the success of automation testing is the determination of test cases. When asked to discuss additional information about the testing strategies used to ensure the reliability of software applications, Organization 8-Participant 2 stated, “test on the edges, test the usual cases that will arise because I guarantee, it will happen when deployed to production (chuckle). I would strongly advocate thorough requirements, documentation, test cases, and test the extremes of whatever you are building.” In a separate study, Hussain, Razak, and Mkpojiogu (2017) found that maximizing automation is an effective way of expediting the testing process. When asked about automation testing, Organization 3-Participant 1 stated,

> I just want to reiterate that in today’s world, when new software is an option, the only way to make sure that you are getting the correct results is to have an environment with automated tests running. While it does not free up QA from performing regression testing, it catches the defect much early on. Therefore, if done correctly, when QA executes the tests, testing should go much smoother.

The benefit of automation testing is that it shortens development cycles, repetitive work, and improves software quality (Garousi & Mantyla, 2016b). Organization 3-Participant 2 reported that their organization strives to do as much automation testing as possible; however, efforts are in place to make testing more efficient using automation. Garousi et al. (2018b) noted that many people think of automation testing only for automated execution of test cases, but automation testing has been successfully implemented in other test-related activities. Deciding when to automate testing is a frequently asked and challenging question (Garousi & Pfahl, 2016). Therefore, choosing a good automation test design determines how testing a particular function or feature should occur.

An analysis of 26 out of the 27 organizational documents included test cases for the requirements to validate that each system workflow would behave adequately. From the literature, De Souza Neto, Moreira, and Musicante (2018) noted that the test design is the process of designing test cases that represent scenarios to be exercised on the system under test. Moreover, Yu, Alegroth, Chatzipetrou, and Gorschek (2020) reported that systems that execute a set of predefined test cases to check regression issues by the latest code changes improve the efficiency and quality of the code. According to Meiliana et al. (2018), automation testing can improve the effort per unit time and the accuracy per test case. Hence, automation testing is nothing but the use of software to perform tests and then determine whether the actual results and predicted results are identical (Eckhart, Meixner, Winkler, & Ekelhart, 2019).

The findings of this study demonstrated that generating test cases for automation testing is in alignment with existing literature. Ping, Xuan, and Xinyue (2017) proposed generating test cases according to test strategies. In research carried out by Zein, Salleh, and Grundy (2016), the authors reported a technique for performing regression testing and the automatic testing of test cases. In another study by Adamsen, Mezzetti, and Moller (2015), the challenge of improving test cases' quality and effectiveness is investigated. The study recognizes the problem of having manually written test cases, not focusing on unusual events. Contributions by Ahmed, Ibrahim, and Ibrahim (2015), which was cited in the professional and academic literature of this study also supported the findings. Ahmed et al. provided a testing approach that addresses the problem of reducing the redundancy in test cases by refactoring source code before test cases are generated. According to Organization 3-Participant 4, “we tend to repeat tests and have the results show up in a database to be verified.” Research carried out by Hamad (2018) reported the benefits of automation testing is that automation is faster, frees up resources for other projects, extensive test coverage, and the technique is more precise with less human error.

When viewed through the unique lens of Lehman’s laws of software evolution, the declining law and the increasing complexity law validates test cases for automation. The declining law states that the quality of evolving software will decline unless the software is strictly maintained, and significant attempts are made to improve it (Lehman, 1996). Meanwhile, Lehman created the increasing complexity law, which states that a software system will become progressively more complex over time unless explicit work is performed to reduce the complexity. In the culture of software testing, assuming that test cases are suitable for automation, the criteria require frequent execution and large volumes of data to perform the repetitive task. Moreover, since the automation of test design and the test script have been advancing well, reducing the effort spent on creating test cases is complex. When asked about the type of projects that are currently being worked on, one participant used the analogy of a job application to illustrate the complexity of automating test cases in a database. The participant stated, “first, you have to have a full and clear understanding of how the database works. All the tables and where the data is moving from one table to another, as you are moving through the front end of the application. If you think of it like a job application, where is the first name, last name, address, things like that? Where does it currently go? Based on the requirements, that will tell me based on the new table data structure and where the data will go."

Lack of automation testing. A significant problem that hinders automated designs is the lack of testing. Testing is significant for software products and plays a vital role in the software development lifecycle. Nearly all of the participants reported a lack of automation testing. Garousi and Mantyla (2016b) explained that the most common obstacle that developers tend to experience when transitioning to an automation-based testing strategy is the lack of testing with automation as most are accustomed to conducting unit tests manually. Software testing is the process of verifying software to find errors (De Souza Neto et al., 2018). Research carried out by De Souza Neto et al. identified the majority of the works dealing with unit tests and the relative lack of automation tools. Kos, Mernik, and Kosar (2016) explained that the lack of automation testing is a result of the lack of experience in developing programming support tools and the belief that high development costs are a contributing factor.

Two organizational documents pointed out the lack of automation testing; in fact, the Execution Summary document stated, "that no automated testing will be performed." From the literature, Garousi and Mantyla (2016b) reported that test automation requires different skills than manual testing. According to Garousi and Mantyla, if the development and testing teams lack programming skills, introducing automation testing to staff requires sufficient training or run the high risk of failure. Moreover, Bruder and

Hasse (2020) noted that a lack of understanding of the automated system leads to overlooking or misinterpreting important information.

The findings in this study demonstrated that the lack of automation testing is in alignment with existing literature. Rabah, Belqasmi, Mizouni, and Dssouli (2016) indicated that the deployment of applications is a costly and complicated process, which is a factor for the lack of automation. In a separate study, Lui et al. (2018) pointed out that commercial software provides sophisticated evaluation tools and fast calculation; however, they lack sufficient robustness to support automation testing.

The conceptual framework that guided this study, Lehman’s laws of software evolution, did not support the findings of this study. Glaser and Strauss (1967) developed a theory titled the grounded theory. The purpose of the grounded theory is to inductively generate theory that is grounded in or emerges from the data. The theory identified three conventional methods used in grounded theory: participant observation, interviewing, and collection of artifacts and texts. Although the lack of automation testing emerged from the data through interviews and organizational documentation, a detailed understanding of the grounded theory is required to make that conclusion.

Unit testing. Unit tests are performed to ensure that a software product is defect-free. Seven participants identified unit testing as a strategy used for testing. One participant agreed that most of the unit testing as far as their organization is concerned is performed by the developers. Another participant added that the first round of testing uses a small set of data, then if confident of the task, more data is added to conduct necessary testing. From the literature, Jan et al. (2016) confirmed that unit testing is performed by software developers using a small set of data. Organization 3-Participant 1 stated, “the vast majority of the testing that I am involved in is unit testing.” Support for this idea also exists in the literature, as Papadakis, Ali, and Perrouin (2019) pointed out that an essential advantage to unit testing is automation. According to Organization 8-Participant 1, “as a developer, we do some unit testing of the code. Our team once conducted manual testing, but we are now writing the scripts for automation.” Alegroth, Feldt, and Kolstrom (2016) found that executing automated unit tests early in the lifecycle identifies defects earlier in the process and is a lot cheaper to fix than those discovered in production. To that end, De Souza Neto et al. (2018) identified unit testing as the verification of one software element in isolation.

An analysis of five out of the 27 organizational documents supported the subtheme of focus on automation testing through unit testing. With that in mind, as part of document analysis, the Unit Testing Design and Best Practices document supported the idea of unit testing, discussing the most prominent best practices for unit testing. From the literature, Yu et al. (2020) reported that the purpose of test automation is to automatically run unit testing, integration testing, performance testing, and user acceptance testing in environments set up for automation tests. In a separate study, Kos et al. (2016) reported that unit testing is usually connected with the Junit testing framework, which is used by developers when implementing unit testing in Java.

Unit tests are designed to make sure that a software product is defect-free. In the culture of software testing, developers perform unit testing to validate code designs. When Lehman (1996) proposed the law of declining quality, he noted that stakeholders would perceive an E-type system that will have declining quality issues unless it is rigorously maintained and adapted to its changing operational environment. Additionally, the law of complexity also supports the subtheme of unit testing because it is implied that the changes required for system evolution makes the system more complex and decreases its quality. Contributions by Shehzad and Shaikh (2017) and Amanatidis and Chatzigeorgeou (2016) validated the process of Lehman's laws suggesting that as the software is developed, frequent changes may be the result of other underlying issues. Organization 3-Participant 1 stated, "a lot of what we do is bug fixes and enhancements on legacy applications where it is virtually impossible to unit test."

Automation tools. Automation testing requires the use of automation tools to reduce human intervention and repeatable tasks. On this point, five participants reported using open source automation tools such as Selenium, Cucumber, and software provided by Kindle. From the literature, Garousi and Mantyla (2016b) and Gojare, Joshi, and Gaigaware (2015) reported that popular open-source tools are often good options as they have low cost and a large user base. Organization 3-Participant 2 stated, "… Selenium is the most common tool you will see from an automation perspective." Previous research carried out by Kalbandi, Pawar, Nikhilkumar, and Bachate (2015) explored automation testing tools such as Quick Time Professional (QTP) and LoadRunner to facilitate the management of automation testing activities. One participant reported that adopting the right automation tool is imperative for test automation since the goal is to make testing easier. To that end, Organization 4-Participant 3 stated, "there could be some better tools utilized, but that goes back to adding more resources to the testing environment."

An analysis of 3 out of the 27 organizational documents supported the subtheme of automation tools. Support for these ideas exist in the literature as Smada, Rotuna, Boneca, and Petre (2018) supported automation software testing. Recent literature further supports the theme focus on automation testing as a strategy software developers can use to ensure the reliability of software applications in the government contracting industry. Neethidevan and Chandraskaran (2018) discussed the benefits of using automation testing to avoid manual effort, while Sharma and Chandra (2019) explored the best technique for testing. Meanwhile, Nidagundi and Novickis (2016) argued that it is always challenging to automate. According to Organization 3-Participant 4, "the thing about testing is that you have to know the nature of the application to know the best technique for testing it." To that end, automation testing reduces software defects, increases productivity, and maximizing organizational profits.

The findings of this study demonstrated that automation tools are in alignment with existing literature. Vila, Novakova, and Todorova (2017) agreed that automation testing uses tools to run tests based on software algorithms to compare the actual outcomes with the expected outcomes. Vila et al. noted that automation testing does not exclude human involvement and the necessity of manual testing because every test case may not meet the conditions to automate. In another study, Yu et al. (2020) reported examples of automation tools, including Junit, Jmeter, and the robot framework. Automation is the most important means for keeping the cost of testing low while guaranteeing an adequate degree of reliability.

The conceptual framework that guided this study, Lehman's laws of software evolution, did not support the findings of this study. Glaser and Strauss (1967) developed a theory titled the grounded theory. The purpose of the grounded theory is to inductively generate theory that is grounded in or emerges from the data. The theory identified three conventional methods used in grounded theory: participant observation, interviewing, and collection of artifacts and texts. Although the discussion of automation tools emerged from the data through interviews and organizational documentation, a detailed understanding of the grounded theory is required to make that conclusion.

Applications to Professional Practice

The specific IT problem that formed the basis of this research was the perceived lack of testing strategies used by software developers to ensure the reliability of software applications in the government contracting industry. Participants in this study provided testing strategies that other software developers could apply to their government-contracted organizations to reduce the risk of software defects. The participants' thoughts on testing strategies spanned from a discussion on unit testing to automation testing. The goal was to identify testing strategies used by software developers in the government contracting industry. There were different opinions on testing strategies used, indicating that a myriad of best practices in the industry applied to various project types in an assortment of ways. After evaluating the collected data, I identified four primary themes associated with software testing strategies: communication and collaboration with all stakeholders, develop well-defined requirements, focus on thorough documentation, and focus on automation testing.

Organizational leaders should consider implementing a software development methodology that maximizes the concept of communication and collaboration with all stakeholders within the team. All of the participants in this research agreed that communication and collaboration with all stakeholders are critical in the culture of software testing as it lays the groundwork for software testing. Equally important, the desired processes should be very flexible and efficient in dealing with change. Communication and collaboration with all stakeholders should enable software developers to design software in stages, making it easier to find and fix software defects. Collaboration allows developers to solve complex problems and learn from each other. Additionally, these processes should be managed throughout the life of the project, coupled with the design, build, and test phases. On this point, a well-defined requirements document should align with organizational expectations resulting in fewer errors, less rework, and an overall improvement in project delivery.

Organizational leaders should establish adopting practices that produce thorough documentation as a mandatory part of their software testing process. The documentation should be clear and concise, generally maintained, and the standard procedure to follow when documenting all policies and procedures. The process of thorough documentation should allow organizational leaders the opportunity to see where there may be inconsistencies or gaps in their current processes. Meanwhile, organizational leaders should promote an open working environment that fosters direct communication. Direct communication should be clear, compelling, and reduces the potential for misunderstanding.

Last, organizational leaders should consider implementing automation testing. While most organizational leaders tend to pass over the aspect of automation testing, the strategy to adopt automation testing increases software efficiency and guarantees robust software quality. All of the participants in this research expressed an interest in automation testing. Organizational leaders can choose to decide the test cases to automate, whether it is those that are more prone to human error or tests that are almost impossible to perform manually. Additionally, there are a host of automation tools that can effectively execute automated test cases without manual intervention; however, organizational leaders can choose which is best for their environment. Participants in the study recommended using tools such as Ranorex, Selenium, Cucumber, Watir, and TestDriveOne for automation testing. These tools can scale tests for complete coverage, prompt reproduction of bugs, perform exploratory tests, and improve software quality. By implementing effective testing practices, organizations can improve software quality that would aid in the deployment of a reliable software application. Cao, Yang, and Liu (2019) noted that software testing is a compelling way to improve and guarantee software reliability and is one of the essential phases in the software development process.

The results obtained from this data could be used as a set of guidelines or best practices for organizations to improve or enhance their current testing processes. The findings of this study may be valuable in professional practice by prompting software developers to consider increasing their knowledge and understanding of effective IT testing processes.

Implications for Social Change

The information from this research may impact social change by providing software developers with the strategies to improve current testing processes. The potential impact of this research is far-reaching for society in general. The benefits of software testing are integral to any project as it allows for the removal of errors and bugs to occur before the release of the product for public use. Whether end-users are accessing websites that provide advice and information about veteran's benefits to healthcare and news and information on NASA space programs to streaming live events occurring at the White House, software testing extends far beyond software development and reaches every area of society. As the dependence on technology grows, a fast pace of change is a warning sign because all changes can lead to risks causing the software to collapse by accident (Hinsen, 2019). A study by Cavalcanti, do Carmo Machado, Anselmo da Motal S. Neto, and Santana de Almeida (2016), found that a change in the software design could impact the responsibility of many developers involved in the project. As noted earlier, communication and collaboration are designed to work together. Collaboration brings people from diverse backgrounds with different perspectives and skillsets together to achieve a common goal. A society exists when people are interacting together, and it is during those activities that people improve their communication and collaboration skills. Therefore, society cannot exist without collaboration.

By adding to the existing body of knowledge, this study's findings may help provide information on testing strategies used to ensure the reliability of software applications. As cited in the professional and academic literature, software errors cost the

U.S. economy an estimated $59.5 billion annually (Chang et al., 2019). This study may be of value to society as its findings may better position organizational leaders for success when considering testing strategies that produce reliable software applications. Lehman's laws of software evolution were vital as it relates to software testing because it highlighted each of the theme areas explored.

This study may also benefit organizational leaders, as well as stakeholders, system analysts, other software developers, software testers, and end-users, because it illustrated testing practices that ensure successful testing of software projects. Critical concerns for most organizations include productivity and profit potential. The knowledge learned from this study could improve software quality, and software testing would receive more emphasis as organizations will need to do more testing to ensure that their products are not vulnerable to defects. As society's dependence on software grows, the knowledge from this study will become more valuable.

Recommendations for Action

The analysis of this study leads to recommendations for action in categories that apply to software developers of government contracting organizations. The study findings revealed an environment that promotes communication and collaboration with all stakeholders as an environment that encourages participation. Organizational leaders should explore adopting testing practices to promote collaboration throughout the software lifecycle. Organizational leaders should use a pre-production testing environment that resembles production to allow for accurate software testing results. I recommend that organizational leaders explore techniques that create formal planning and peer code review processes during all phases of the software lifecycle. Accurate documentation should flow from well-defined requirements and should be the baseline that explains why the features exist and why specific corrective actions form defects.

Moreover, organizational leaders should explore automated software testing as a viable option because it saves time and money, improves software quality, and expands test coverage. Automated testing should be considered as a testing strategy to speed up test execution and to reduce redundant test cases. To that end, test cases should be well documented and executed by multiple resources as it will improve software testing knowledge and growth skills. Furthermore, automation testing has enabled organizations to accomplish more activities with higher productivity, significantly increases the functionality of the software application portfolio, and minimizes the effect of platform changes.

This study may also benefit organizational leaders, as well as stakeholders, system analysts, other software developers, software testers, and end-users because it illustrated testing best practices that ensure successful testing of software projects. Once the study is approved, I will disseminate the results of the literature through conferences, scholarly journals, business journals, and training. Furthermore, copies of the final study will be provided via email to all study stakeholders and participants.

Recommendations for Further Study

The purpose of this qualitative multiple case study was to explore the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. The population for the study consisted of software developers of three government-contracted organizations who agreed to participate in this study.

My recommendations for further research derive from the limitations related to the research, from the literature, and from the information obtained while conducting interviews. The first limitation of the study included a potential for bias and preconceived ideas and values because the results are subjective and may result in research limitations. I recommend additional research to be conducted using a different design or method. For this reason, a quantitative study may examine the correlation of the results. I also recommend that additional qualitative research studies include other organizations, industries, and locations to determine whether the findings from new research would correspond to my findings. Another recommendation would be to explore the perceptions of quality assurance engineers, software testers, and others involved in software development projects. Besides, future studies should incorporate the effect of pair programming to determine if it is a significant factor for efficient exploratory testing.

The second limitation of the study referenced participants responding to interview questions based on what they believe the interviewer wants to hear. The participants shared information regarding current processes and detailed information on why those processes and procedures are in place. Therefore, researchers have the capability of expanding on this research by broadening the scope of participants outside of the software development team. For this reason, the researcher can explore the same research with other groups to understand how testing strategies play a role in their processes. In the end, this study has contributed to the literature and paved the way for additional research in the IT industry.

Reflections

The doctoral research study was one of the most challenging academic decisions I have ever experienced. Consequently, this study changed my perception of academic research and intensified my admiration for those pursuing or have pursued the highest level of academia that one can achieve. When I began the study, I did not realize that there would be long days and nights ahead with few weekends to spare. For me, the doctoral study was a journey filled with obstacles, but also enlightenment. Each time that I encountered an obstacle, even if it was challenging, I persevered harder and expanded my knowledge on the process. I made several essential notes while on my DIT doctoral journey. First, writing the doctoral study calls for discipline and stamina. Second, finding organizations to participate in my research was another obstacle that I found challenging. From the government shutdown to organizations choosing not to participate was daunting, but the reality of the fact was I continued to persevere. I did not know any of the potential participants or organizations to avoid any bias. The positive outcome was that participants from government contracting organizations were willing to share their knowledge and experiences. Third, I followed the interview protocols and bracketing techniques to mitigate potential bias or preconceived ideas and values to ensure the credibility of the study. In the end, I gained an understanding of the qualitative research method and case study design. I learned how to conduct academic research, how to analyze data, and explain how research affects others.

Summary and Study Conclusions

The purpose of this qualitative multiple case study was to explore the testing strategies software developers use to ensure the reliability of software applications in the government contracting industry. In my efforts to reveal strategies that software developers could use to ensure the reliability of software applications in the government contracting industry, altogether I hope that I have provided information on the importance of developing well-defined requirements, thorough documentation, and automation testing in my research on the topic. The constant need for effective policies and procedures in the area of software testing is all about meeting the needs and expectations of the customer concerning the design, functionality, and reliability of software.

Software testing is a skill and a complex process that involves creativity, experience, and intuition. It is more than a bug hunting activity, but a process used to verify and validate requirements and specifications. The process starts with the creation of well-defined requirements and requires communication and collaboration with all stakeholders. The process is thoroughly documented, providing a blueprint for future software development work. A well-ordered and structured process can significantly improve the efficiency and effectiveness of routine tasks.

Organizations make significant investments into their software systems, which are crucial to business assets. While maintaining the value of business assets, the software must be verified and thoroughly tested as a cost-effective means of implementing a quality product. Recent studies showed that many organizations are starting to automate their testing processes to save money and improve quality. As the dependence on today's technology grows, society is more invested in the quality and reliability of software testing. Thus, spreading the awareness of software testing is beneficial to consumers, businesses, and governments to foster a better understanding of the processes.

References

Aagard, J. (2017). Introducing post phenomenological research: A brief and selective sketch of phenomenological research methods. *International Journal of Qualitative Studies in Education, 30*(6), 519-533. doi:10.1080/09518398.2016.1263884

Aceituna, D., & Do, H. (2019). Addressing the state explosion problem when visualizing off-nominal behaviors in a set of reactive requirements. *Requirements Engineering, 24*(2), 161-180. doi:10.1007/s00766-017-0281-y

Adamsen, C. Q., Mezzetti, G., & Moller, A. (2015). Systematic execution of android test suites in adverse conditions. *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ACM,* 83-93. doi:10.1145/2771783.2771786

Adnan, N. F., & Ritzhaupt, A. D. (2018). Software engineering design principles applied to instructional design: What can we learn from our sister discipline? *TechTrends*, *62*, 77-94. doi:10.1007/s11528-017-0238-5

Afzal, W., Alone, S., Glocksien, K., & Torkar, R. (2016). Software test process improvement approaches: A systematic literature review and an industrial case study. *The Journal of Systems and Software*, *111*, 1-33. doi:10.1016/j.jss.2015.08.048

Ahmad, T., Truscan, D., & Porres, I. (2018). Identifying worst-case user scenarios for performance testing of web applications using markov-chain workload models. *Future Generation Computer Systems, 84*, 910-920. doi:10.1016/j.future.2018.01.042

Ahmed, B. S., Abdulsamad, T. S., & Potrus, M. Y. (2015). Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm, *Information and Software Technology*, *66*, 13-29. doi:10.1016/j.infsof.2015.05.005

Ahmed, M., Ibrahim, R., & Ibrahim, N. (2015). An adaption model for android application testing with refactoring. *International Journal of Software Engineering & Applications*, *9*(10), 65-74. Retrieved from http://www.airccse.org/journal/ijsea/ijsea

Alahyari, H., Gorschek, T., & Svensson, R. B. (2019). An exploratory study of waste in software development organizations using agile or lean approaches: A multiple case study at 14 organizations. *Information and Software Technology*, *105*, 78-94. doi:10.1016/j.infsof.2018.08.006

Alahyari, H., Svensson, R. B., & Gorschek, T. (2017). A study of value in agile software development organizations. *The Journal of Systems and Software*, *125*, 271-288. doi:10.1016/j.jss.2016.12.007

Al-Dhaafri, H. S., & Al-Swidi, A. (2016). The impact of total quality management and entrepreneurial orientation on organizational performance. *International Journal of Quality & Reliability Management*, *33*(5), 597-614. doi:10.1108/IJQRM-03-2014-0034

Alegroth, E., Feldt, R., & Kolstrom, P. (2016). Maintenance of automated test suites in industry: An empirical study on visual gui testing. *Information and Software Technology*, *73*, 66-80. doi:10.1016/j.infsof.2016.01.012

Alenezi, M., & Almustafa, K. (2015). Empirical analysis of the complexity evolution in open-source software systems. *International Journal of Hybrid Information Technology*, *8*(2), 257-266. doi:10.14257/ijhit.2015.8.2.24

Alhammad, M. M., & Moreno, A. M. (2018). Gamification in software engineering education: A systematic mapping. *Journal of Systems and Software, 141*, 131-150. doi:10.1016/j.jss.2018.03.065

Ali, N., & Lai, R. (2016). A method of requirements change management for global software development. *Information and Software Technology, 70,* 49-67. doi:10.1016/j.infsof.2015.09.005

Allison, M., & Joo, S. F. (2015). An adaptive delivery strategy for teaching software testing and maintenance. *2015 10th International Conference on Computer Science & Education*, *IEEE,* 237-242. doi:10.1109/ICCSE.2015.7250249

Almugrin, S., Albattah, W., & Melton, A. (2016). Using indirect coupling metrics to predict package maintainability and testability. *Journal of Systems and Software, 121*, 298-310. doi:10.1016/j.jss.2016.02.024

Alnabhan, M., Hammouri, A., Hammod, M., Atoum, M., & Al-thnebat, O. (2018). 2d visualization for object-oriented software systems. *2018 International Conference on Intelligent Systems and Computer Vision, IEEE*. doi:10.1109/ISACV.2018.8354085

Alsaqaf, W., Daneva, M., & Wieringa, R. (2019). Quality requirements challenges in the context of large-scale distributed agile: An empirical study. *Information and Software Technology, 110,* 39-55. doi:10.1016/j.infsof.2019.01.009

Alvaro, P., & Tymon, S. (2017). Abstracting the geniuses away from failure testing. *Queue*, *15*(5), 10. doi:10.1145/3155112.3155114

Alzoubi, Y. I., Gill, A. Q., & Al-Ani, A. (2016). Empirical studies of geographically distributed agile development communication challenges: A systematic review. *Information & Management*, *53*, 22-37. doi:10.1016/j.im.2015.08.003

Amanatidis, T., & Chatzigeorgeou, A. (2016). Studying the evolution of php web applications. *Information & Software Technology*, *72*, 48-67. doi:10.1016/j.infsof.2015.11.009

Ammann, P., & Offutt, J. (2016). *Introduction to software testing*. Cambridge University Press.

Anand, A., Singh, O., & Das, S. (2015). Fault severity based multi up-gradation modeling considering testing and operational profile. *International Journal of Computer Applications*, *124*(4), 9-15. Retrieved from http://www.ijcaonline.org

Anderson-Cook, C. M., Lu, L., & Parker, P. A. (2019). Effective interdisciplinary collaboration between statisticians and other subject matter experts. *Quality Engineering*, *31*, 164-176. doi:10.1080/08982112.2018.1530357

Andrews, A., Alhaddad, A., & Boukhris, S. (2019). Black-box model-based regression testing of fail-safe behavior in web applications. *The Journal of Systems & Software*, *149*, 318-339. doi:10.1016/j.jss.2018.11.020

Andrews, A., Elakeili, S., & Alhaddad, A. (2015). Selective regression testing of safety-control systems: A black box approach. *2015 IEEE International Conference on*

Software Quality, Reliability, and Security-Companion, IEEE. 22-31. doi:10.1109/QRS-C.2015.16

Anthonisz, S., & Perry, C. (2015). Effective marketing of high-rise luxury condominiums in a middle-income country like Sri Lanka. *Journal of Work-Applied Management, 7*, 61-83. doi:10.1108/jwam-10-2015-002

Anthopoulos, L., Reddick, C. G., Giannakidou, I., & Mavridis, N. (2016). Why e-government projects fail? An analysis of the healthcare.gov website. *Government Information Quarterly, 33*,161-173. doi:10.1016/j.giq.2015.07.003

Antinyan, V., & Staron, M. (2017). Rendex: A method for automated reviews of textual requirements. *Journal of Systems and Software*, *131*, 63-77. doi:10.1016/j.jss.2017.05.079

Anu, V., Hu, W., Carver, J. C., Walia, G. S., & Bradshaw, G. (2018). Development of a human error taxonomy for software requirements: A systematic literature review. *Information and Software Technology*, *103,* 112-124. doi:10.1016/j.infsof.2018.06.011

Aovak, A., Gugan, K., Varga, M., & Domotor, A. (2018). Creation of an annotated corpus of old and middle Hungarian court records and private correspondence. *Language Resources and Evaluation*, *52*,1-28. doi:10.1007/s10579-017-9393-8

Arif-Uz-Zaman, K., Cholette, M. E., Ma, L., & Karim, A. (2017). Extracting failure time data from industrial maintenance records using text mining. *Advanced Engineering Informatics, 33,* 388-396. doi:10.1016/j.aei.2016.11.004

Arora, P. K., & Bhatia, R. (2018). A systematic review of agent-based test case generation for regression testing. *Arabian Journal for Science and Engineering, 43*(2), 447-470. doi:10.1007/s13369-017-2796-4

Arundell, F., Mannix, J., Sheehan, A., & Peters, K. (2018). Workplace culture and the practice experience of midwifery students: A meta-synthesis. *Journal of Nursing Management*, *26*(3), 302-313. doi:10.1111/jonm.12548

Ashmore, S., Townsend, A., Demarie, S., & Mennecke, B. (2018). An exploratory examination of modes of interaction and work in waterfall and agile teams. *International Journal of Agile Systems and Management*, *11*, 67-102. doi:10.1504/IJASM.2018.091361

Aversano, L., Di Brino, M., Guardabascio, D., Salerno, M., & Tortorella, M. (2015). Understanding enterprise open source software evolution. *Procedia Computer Science*, *64*, 924-931. doi:10.1016/j.procs.2015.08.609

Badenhorst, C. (2018). Citation practices of postgraduate students writing literature reviews. *London Review of Education*, *16*, 121-135. doi:10.18546/LRE.16.1.11

Badri, P., Wolfe, R., Farmer, A., & Amin, M. (2018). Psychosocial determinants of adherence to preventive dental attendance for preschool children among filipino immigrants in Edmonton, Alberta. *Journal of Immigrant and Minority Health*, *20*(3), 658-667. doi:/10.1007/s10903-017-0599-z

Bagheri, H., Garcia, J., Sadeghi, A., Malek, S., & Medvidovic, N. (2016). Software architectural principles in contemporary mobile software: From conception to practice. *Journal of Systems and Software*, *119*, 31-44. doi:10.1016/j.jss.2016.05.039

Bagherzadeh, M., Kahani, N., Bezemer, C. P., Hassan, A. E., Dingel, J., & Cordy, J. R. (2018). Analyzing a decade of linux system calls. *Empirical Software Engineering*, *23*(3), 1519-1551. doi:10.1007/s10664-017-9551-z

Bahamdain, S. S. (2015). Open source software (oss) quality assurance: A survey paper. *Procedia Computer Science*, *56,* 459-464. doi:10.1016/j.procs.2015/07.236

Bansal, M. S., Kellis, M., Kordi, M., & Kundu, S. (2018). Ranger-DTL 2.0: Rigorous reconstruction of gene-family evolution by duplication, transfer and loss. *Bioinformatics*, *34*(18), 3214-3216. doi:10.1093/bioinformatics/bty314

Barnes, J. (2015). Qualitative research from start to finish (2nd Edition). *Neuropsychological Rehabilitation*, *27*(8), 1156-1158. doi:10.1080/09602011.2015.1126911

Barnham, C. (2016). Quantitative and qualitative research: Perceptual foundations. *International Journal of Market Research*, *57,* 837–854. doi:10.2501/ijmr-2015

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, *41*(5), 507-525. Retrieved from https:// ieeexplore.ieee.org/

Bartels, D. M., Hastie, R., & Urminsky, O. (2018). Connecting laboratory and field research in judgement and decision making: Causality and the breadth of external validity. *Journal of Applied Research in Memory and Cognition, 7*, 11-15. doi:10.1016/j.jarmac.2018.01.001

Baskarada, S. (2014). Qualitative case study guidelines. *The Qualitative Report*, *19*(40), 1-25. Retrieved from http://nsuworks.nova.edu/tqr/

Batarseh, F. A., & Gonzalez, A. J. (2015). Validation of knowledge-based systems: A reassessment of the field. *Artificial Intelligence Review*, *43*(4), 485-500. doi:10.1007/s10462-013-9396-9

Batool, A. (2015). A comprehensive analysis on software testing methodologies. *International Journal of Computer and Communication System Engineering* (*IJCCSE*), *2*(3), 387-392. Retrieved from http://www.ijccse.com

Becher, E. H., & Wieling, E. (2015). The intersections of culture and power in clinician and interpreter relationships: A qualitative study. *Cultural Diversity and Ethnic Minority Psychology*, *21*, 450-457. doi:10.1037/a0037535

Belady, L. A., & Lehman, M. M. (1976). A model of large program development. *IBM Systems Journal*, *15*(3), 225-252. doi:10.1147/sj.153.0225

Bell, J. S., Murray, F. Z., & Davies, E. L. (2019). An investigation of the features facilitating effective collaboration between public health experts and data scientists at a hackathon. *Public Health*, *173*, 120-125. doi:10.1016/j.puhe.2019.05.007

Beller, M. (2018). Toward an empirical theory of feedback-driven development. *2018 IEEE/ACM 40th International Conference on Software Engineering, IEEE*, 503-505. doi:10.1145/3183440.3190332

Bellery, L., Hodges, H., Camp, A., & Aduddell, K. (2016). Teamwork in acute care: Perceptions of essential but unheard assistive personnel and the counterpoint of perceptions of registered nurses. *Research in Nursing & Health*, *39*(5), 337-346. doi:10.1002/nur.21737

Beppe, T. A., de Sousa-Santos, I., Linhares de Araujo, I., Aragao, B. S., Ximenes, D., & Andrade, R. M. C. (2018). Greatest: A card game to motivate the software testing learning. *SBES '18: Proceedings of the XXXII Brazilian Symposium on Software Engineering, ACM*. 298-307. doi:10.1145/3266237.3266254

Berger, R. (2015). Now i see it, now i don’t: Researcher’s position and reflexivity in qualitative research. *Qualitative Research*, *15*, 219-234. doi:10.1177/1468794112468475

Bergmane, L., Grabis, J., & Zeiris, E. (2017). A case study: Software defect root causes. *Information Technology and Management Science*, *20*, 54-57. doi:10.1515/itms-2017-0009

Berman, A. C., & Chutka, D. S. (2016). Assessing effective physician-patient communication skills: “Are you listening to me, doc?” *Korean Journal Medical Education*, *28*(2), 243-249. doi:10.3946/kjme.2016.21

Beskow, L. M., Check, D. K., & Ammarell, N. (2014). Research participants’ understanding of and reactions to certificates of confidentiality. *AJOB Primary Research, 5*, 12-22. doi:10.1080/21507716.2013.813596

Beverly, E. A., Hamel-Lambert, J., Jensen, L. L., Meeks, S., & Rubin, A. (2018). A qualitative process evaluation of a diabetes navigation program embedded in an endocrine specialty center in rural appalachian ohio. *BMC Endocrine Disorders*, *18*, 1-15. doi:10.1186/s12902-018-0278-7

Bian, Y., Parande, M. A., Koru, G., & Zhao, S. (2016). Testing the theory of relative dependency from an evolutionary perspective: higher dependencies concentration in smaller modules over the lifetime of software products. *Journal of Software: Evolution and Process*, *28*(5), 340-371. doi:10.1002/smr.1774

Biros, M. (2018). Capacity, vulnerability, and informed consent for research. *The Journal of Law, Medicine, & Ethics*, *46*, 72-78. doi:10.11771107310518766021

Blackmon, S. (2017). [Insert emoji or avatar here]: Phenomenology and digital research. *Journal of Gaming & Virtual Worlds*, *9*(3), 193-205. doi:10.1386/jgvw.9.3.193_1

Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*, 64–69. doi:10.1109/2.976920

Bonello, M., & Meehan, B. M. (2019). Transparency and coherence in a doctoral study case analysis: Reflecting on the use of nvivo within a framework approach. *The Qualitative Report*, *24*(3), 483-498. Retrieved from https://nsuworks.nova.edu/tqr

Borgers, A., Pownall, R., & Raes, L. (2016). Acquaintance networks and attitudes to climate change. *Academic Homepage Louis Raes*, 1-12. Retrieved from http://www.louisraes.com

Bouras, Z. E., & Maouche, M. (2017). Software architectures evolution-based merging. *Informatica*, *41*, 111-120. Retrieved from http://www.informatica.si

Bowden, C., & Galindo-Gonzalez, S. (2015). Interviewing when you’re not face-to-face: The use of email interviews in a phenomenological study. *International Journal of Doctoral Studies*, *10*, 79-92. Retrieved from http://www.informingscience.org

Braga de Vasconcelos, J., Kimble, C., Carreteiro, P., & Rocha, A. (2017). The application of knowledge management to software evolution. *International Journal of Information Management*, *37,* 1499-1506. doi:10.1016/j.ijinfomgt.2016.05.005

Brennan, P. F., & Bakken, S. (2015). Nursing needs big data and big data needs nursing. *Journal of Nursing Scholarship*, *47*, 477-484. doi:10.1111/jnu.12159

Brhel, M., Meth, H., Maedche, A., & Werder, K. (2015). Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, *61*, 163-181. doi:10.1016/j.infsof.2015.01.004

Brichni, M., Dupuy-Chessa, S., Gzara, L., Mandran, N., & Jeannet, C. (2017). BI4BI: A continuous evaluation for business intelligence systems. *Expert Systems with Applications*, *76*, 97-112. doi:10.1016/j.eswa.2017.01.018

Britto, R., Mendes, E., & Borstler, J. (2015). An empirical investigation on effort estimation in agile global software development. *2015 IEEE 10th International Conference on Global Software Engineering, IEEE*, 38-45. doi:10.1109/ICGSE.2015.10

Bronckers, L. A., Roc'h, A., & Smolders, A. B. (2017). How tough are the front-end requirements for 4g-and-beyond handsets? *2017 4th European Microwave Conference, IEEE,* 711-714. doi:10.23919/EUMC.2017.8230946

Bruder, C., & Hasse, C. (2020). What the eyes reveal: Investigating the detection of automation failures. *Applied Ergonomics, 82,* 1-10. doi:10.1016/j.apergo.2019.102967

Bruyn, P. D., Mannaert, H., Verelst, J., & Huysmans, P. (2018). Enabling normalized systems in practice-exploring a modeling approach. *Business & Information Systems Engineering*, 60, 55-67. doi:10.1007/s12599-017-0510-4

Buckley, I., & Buckley, W. (2017). Teaching software testing using data structures. *International Journal of Advanced Computer Science and Applications*, *8*(4). 1-4. Retrieved from http://www.thesai.org

Budde, K., Zimmerman, J., Neuhaus, E., Schroder, M., Uhrmacher, A. M., & van Rienen, U. (2019). Requirements for documenting electrical cell stimulation experiments for replicability and numerical modeling. *2019 41st Annual International Conference of IEEE Engineering in Medicine & Biology Society, IEEE*, 1082-1088. doi:10.1109/EMBC.2019.8856863

Burman, E., Hansbo, P., & Larson, M. (2018). A cut finite element method with boundary value correction mathematics of computation. *American Mathematical Society*, *87*, 633-657. doi:10.1090/MCOM/3240

Cacari-Stone, L., Wallerstein, N. G., & Minkler, M. (2014). The promise of community based participatory research for health equity: A conceptual model for bridging evidence with policy. *American Journal of Public Health*, *104*, 1615-1623. doi:10.2105/AJPH.2014.301961

Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., … & Rodriguez, D. (2015). Moving fast with software verification. *NASA Formal Methods*, 3-11. doi:10.1007/978-3-319-17524-9_1

Caliz, D., Samaniego, G., & Caliz, R. (2016). Methodological proposal of policies and procedures for quality assurance in information systems for software development companies based on CMMI. *Journal of Software, 11*(3), 230-241. doi:10.17706/jsw.11.3.230-241

Camilo, J. R. M., L'erario, A., Pagotto, T., & Fabri, J. A. (2018). A process for distributed software evolution: A proprietary software case study. In *Proceedings of the 13th Conference on Global Software Engineering, ACM*, 44-53. doi:10.1145/3196369.3196376

Campanelli, A. S., Camilo, R. D., & Parreiras, F. S. (2018). The impact of tailoring criteria on agile practices adoption: A survey with novice agile practitioners in Brazil. *The Journal of Systems and Software*, *137*, 366-379. doi:10.1016/j.jss.2017.12.012

Canli, S., & Demirtaş, H. (2018). The effects of school principals' trust in teachers on school climate. *International Online Journal of Educational Sciences, 10*, 105. doi:10.15345/iojes.2018.01.010

Cao, P., Yang, K., & Liu, K. (2019). Optimal selection and release problem in software testing process: A continuous time stochastic control approach. *European Journal of Operational Research.* doi:10.1016/j.ejor.2019.01.075

Capgemini Group (2016). World quality report 2015-16 tech report. Retrieved from http://www.capgemini.com/though-leadership/world-quality-report-2015-16

Capili, R. D. (2018). Using system dynamics modeling to understand the impact of federal constraints in implementing the agile methodology within the United

States federal government. (Doctoral dissertation, George Washington University). Proquest id:10743636

Caretta, M. A. (2016). Member checking: A feminist participatory analysis of the use of preliminary results pamphlets in cross-cultural, cross-language research. *Qualitative Research*, *16*(3), 305-318. doi:10.1177/1468794115606495

Carter, N., Bryant-Lukosius, D., DiCenso, A., Blythe, J., & Neville, A. J. (2014). The use of triangulation in qualitative research. *Oncology Nursing Forum*, *41*, 545-547. doi:10.1188/14.ONF.545-547

Cashman, S., & Rosenblatt, H. J. (2014). *Systems Analysis and Design* (10th ed). Boston, MA: Cengage Technology

Cavalcanti, Y. C., do Carmo Machado, I., Anselmo da Motal S. Neto, P., & Santana de Almeida, S. (2016). Towards semi-automated assignment of software change requests. *Journal of Systems and Software*, *115*, 82-101. doi:10.1016/j.jss.2016.01.038

Cazals, F., & Dreyfus, T. (2017). The structural bioinformatics library: Modeling in biomolecular science and beyond. *Bioinformatics*, *33*(7), 997-1004. doi:10.1093/bioinformatics/btw752

Chandani, P., & Gupta, C. (2018). An exhaustive requirement analysis approach to estimate risk using requirement defect and execution flow dependency for software development. *Journal of Information Technology Research*, *11*(2), 68-87. doi:10.4018/jitr.2018040105

Chandraprabha, C., Kumar, A., & Saxena, S. (2015). Data driven testing framework using selenium web driver. *International Journal of Computer Applications, 118*(18), 18-23. doi:10.5120/20845-3497

Chang, S. A., Seow, G., & Tam, K. (2019). Debugging debugging. *Journal of Multidisciplinary Research*, *11*, 51-64. Retrieved from www.jmrpublication.org

Chari, K., & Agrawal, M. (2018). Impact of incorrect and new requirements on waterfall software project outcomes. *Empirical Software Engineering, 23*, 165-185. doi:10.1007/s10664-017-9506-4

Charman, A. J., Petersen, L. M., Piper, L. E., Liedeman, R., & Legg, T. (2015). Small area census approach to measure the township informal economy in South Africa. *Journal of Mixed Methods Research*, *11*, 36-58. doi:10.1177/1558689815572024

Charrada, E. B., Koziolek, A., & Glinz, M. (2015). Supporting requirements update during software evolution. *Journal of Software: Evolution and Process*, *27*(3) 166-194. doi:10.1002/smr.1705

Chen, L. H., Wu, C. H., Lin, S. H., & Ye, Y. C. (2018). Top down or bottom up? The reciprocal longitudinal relationship between athletes' team satisfaction and life satisfaction. *Sport, Exercise, and Performance Psychology*, *7*, 1-12. doi:10.1037/spy0000086

Chen, S. M., & Han, W. H. (2018). A new multiattribute decision making method based on multiplication operations of interval-valued intuitionistic fuzzy values and linear programming methodology. *Information, Sciences, 429*, 421-432. doi:10.1016/j.ins.2017.11.018

Chen, T. H., Shang, W., Nagappan, M., Hassan, A. E., & Thomas, S. W. (2017). Topic-based software defect explanation. *Journal of Systems and Software*, *129*, 79-106. doi:10.1016/j.jss.2016.05.015

Chen, T. Y., Kuo, F. C., Towey, D., & Zhou, Z. Q. (2015). A revisit of three studies related to random testing, *Science China*. *Information Sciences*, *58*(5), 1-9. doi:10.1007/s11432-015-5314-x

Chiumento, A., Khan, M. N., Rahman, A., & Frith, L. (2015). Managing ethical challenges to mental health research in post conflict settings. *Developing World Bioethics*. doi:10.1111/dewb.12076

Clarke, P. J., Davis, D., King, T. M., Pava, J., & Jones, E. (2014). Integrating testing into software engineering courses supported by a collaborative learning environment. *ACM Transactions on Computing Education*, *14*(3), 18:1-18:33. doi:10.1145/2648787

Clarke, V., & Braun, V. (2018). Using thematic analysis in counselling and psychotherapy research: A critical reflection. *Counselling and Psychotherapy Research*, *18*(2), 107-110. doi:10.1002/capr.12165

Coelho, J., Valente, M. T., Silva, L. L., & Shihab, E. (2018). Identifying unmaintained projects in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ACM*, 15. doi:10.1145/3239235.3240501

Coenen, M., Cabello, M., Umlauf, S., Ayuso-Mateos, J. L., Anczewska, M., & Leonardi, M. (2016). Psychosocial difficulties from the perspective of persons with neuropsychiatric disorders. *Disability and Rehabilitation*, *38*(12), 1134-1145. doi:10.3109/09638288.2015.1074729

Collette, A. E., Wann, K., Nevin, M. L., Rique, K., Tarrant, G., Hickey, L. A., … Thomason, T. (2017). An exploration of nurse-physician perceptions of collaborative behaviour. *Journal of Interprofessional Care*, *31*(14), 470-478. doi:10.1080/13561820.2017.1301411

Cooper, R. G., & Sommer, A. F. (2018). Agile-stage gate for manufacturers. Changing the way new products are developed integrating agile project management methods into a stage-gate system offers both opportunities and challenges. *Research-Technology Management*, *61*(2), 17-26. doi:10.1080/08956308.2018.1421380

Cope, D. G. (2014). Methods and Meanings: Credibility and trustworthiness of qualitative research. *Oncology Nursing Forum*, *41*, 89-91. doi:10.1188/14.ONF.89-91

Craft, A. R. (2019). Online documentation portals in library technical services: Shedding light on local practices and procedures. *Serials Review*, *45*(3), 171-175. doi:10.1080/00987913.2019.1645531

Crevier, L. P., & Parrott, L. (2019). Synergy between adaptive management and participatory modelling: The two processes as interconnected spirals. *Ecological Informatics, 53*, 1-6. doi:10.1016/j.ecoinf.2019.100982

Cronin, C. (2014). Using case study research as a rigorous form of inquiry. Nurse Researcher, *21*(5), 19-27. doi:10.7748/nr.21.5.19.e1240

Crowe, M., Inder, M., & Porter, R. (2015). Conducting qualitative research in mental health: Thematic and content analysis. *Australian & New Zealand Journal of Psychiatry*, *49*, 616-623. doi:10.1177/0004867415582053

Cruzes, D. S., Moe, N. B., & Dybå, T. (2016). Communication between developers and testers in distributed continuous agile testing. In *Global Software Engineering (ICGSE), 2016 IEEE 11th International Conference, IEEE*, 59-68. doi:10.1109/ICGSE.2016.27

Cuellar, A. M. (2018). Self-reflexivity through journaling: An imperative process for the practicing clinician. *The William and Mary Educational Review*, *5*, 69-82. Retrieved from https://scholarworks.wm.edu/wmer/

Cundiff, J., McCallum, T., Rich, A., Truax, M., & Ward, T. (2015). Healthcare.gov: Opportunity out of disaster: Teaching case. *Journal of Information Systems Education*, *25*(4), 289-293. Retrieved from www.jise.org

Curcio, K., Navarro, T., Malucelli, A., & Reineher, S. (2018). Requirements engineering: A systematic mapping study in agile software development. *The Journal of Systems and Software*, *139*, 32-50. doi:10.1016/j.jss.2018.01.036

Dadkhah, M., Araban, S., & Paydar, S. (2020). A systematic literature review on semantic web enabled software testing. *The Journal of Systems and Software*, *162*, 1-34. doi:10.1016/j.jss.2019.110485

Dalal, S., & Hooda, S. (2018) Aspect-oriented software testing techniques: A review. *International Journal of Advanced Research in Computer Science*, *9*(2), 211-216. www.ijarcs.info/index.php/ijarcs

Dalal, S., & Solanki, K. (2018). Challenges of regression testing: A pragmatic perspective. *International Journal of Advanced Research in Computer Science*, *9*, 499-503. Retrieved from www.ijarcs.info

Daniel, B. K. (2018). Empirical verification of the "TACT" framework for teaching rigor in qualitative research methodology. *Qualitative Research Journal*, *18*(3), 262-275. doi:10.1108/QRJ-D-17-00012

Daniel, J. (2014). *Sampling essentials: Practical guidelines for making sampling choices*. Los Angeles, CA: Sage.

Dasan, S., Gohil, P., Cornelius, V., & Taylor, C. (2015). Prevalence causes and consequences of compassion satisfaction and compassion fatigue in emergency care: a mixed-methods study of UK NHS Consultants. *Emergency Medicine Journal*, *32*(8). 588-594. doi:10.1136/emermed-2014-203671

Dasgupta, M. (2015). Exploring the relevance of case study research. *Vision* (09722629), *19*(2). 147-160. doi:10.1177/0972262915575661

David, S. L., & Hitchcock, J. H. (2018). Understanding patient trust in the athletic setting through interviews. *The Internet Journal of Allied Health Sciences and Practice*, *16*(2), 4. Retrieved from https://nsuworks.nova.edu/ijahsp

Deak, A., Stålhane, T., & Sindre, G. (2016). Challenges and strategies for motivating software testing personnel. *Information and Software Technology*, 731-15. doi:10.1016/j.infsof.2016.01.002

Decan, A., Mens, T., & Grosjean, P. (2018). An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*, 1-36. doi:10.1007/s10664-017-9589-y.

DeGirolamo, A. M., Di Pillo, R., Porto, A. L., Todisco, M. T., & Barca, E. (2018). Identifying a reliable method for estimating suspended sediment load in a temporary river system. *Catena*, *165*, 442-453. doi:10.1016/j.catena.2018.02.015

De Loo, I., Cooper, S., & Manochin, M. (2015). Enhancing the transparency of accounting research. The case of narrative analysis. *Qualitative Research in Accounting & Management*, *12*, 34-54. doi:10.1108/QRAM-02-2013-0007

De Massis, A., & Kotlar, J. (2014). The case study method in family business research: Guidelines for qualitative scholarship. *Journal of Family Business Strategy*, *5*, 15-29. doi:10.1016/j.jfbs.2014.01.007

Deng, J. (1982). Grey systems control. *Systems & Control Letters*, *1*, 288-294. doi:10.1016/S0167-6911(82)80025-X

Dennis, V., & Walcott, J. (2014). Federal financial management shared services: The move is on. *The Journal of Government Financial Management*, *63*(3), 18-24. Retrieved from http://www.agacgfm.org/Resources/Journal-of-Government-Financial-Management.aspx

Derobertmasure, A., & Robertson, J. E. (2014). Data analysis in the context of teacher training: Code sequence analysis using qda miner. *Quality and Quantity*, *48*(4), 2255-2276. doi:10.1007/s11135-013-9890-9

De Souza, E. F., De Almeida Falbo, R., & Vijaykuman, N. L. (2015). Knowledge management initiatives in software testing: A mapping study. *Information and Software Technology, 57,* 378-391. doi:10.1016/j.infsof.2014.05.016

De Souza Neto, J. B., Moreira, A. M., & Musicante, M. A. (2018). Semantic web services testing: A systematic mapping study. *Computer Science Review*, *28,* 140-156. doi:10.1016/j.cosrev.2018.03.002

Despa, M. L. (2015). Formulating the isdf software development methodology. *Informatica Economica*, *19,* 66-80. doi:10.12948/issn14531305/19.2.2015.07

Dhandapani, S. (2016). Integration of user-centered design and software development process. *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference, IEEE,* 1-5. doi:10.1109/IEMCON.2016.7746075

Di Alesio, S., Briand, L. C., Nejati, S., & Gotlieb, A. (2015). Combining genetic algorithms and constraint programming to support stress testing of task deadlines. *ACM Transactions on Software Engineering and Methodology*, *25,* 4:1-4:37. doi:10.1145/2818640

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *The Journal of Systems and Software*, *119,* 87-108. doi:10.1016/j.jss.2016.06.013

Dolezel, M., & Buchalcevova, A. (2015). Test governance framework for contracted is development: ethnographically information action research. *Information and Software Technology*, *65*, 69-94. doi:10.1016/j.infsof.2015.03.003

Dosemagen, S., Liboiron, M., & Molly, J. (2016). Gathering for open science hardware 2016. *Journal of Open Hardware, 1*, 4. doi:10.5334/joh.5

Dos Santos-Neto, B. F., Ribeiro, M., Da Silva, V. T., Braga, C., de Lucena, C. J. P., & de Barros Costa, E. (2015). AutoRefactoring: A platform to build refactoring agents. *Expert Systems with Applications*, *42*(3), 1652-1664. doi:10.1016/j.eswa.2014.09.022

Dou, E. (2016). Cost model for verifying requirements. 2016 IEEE AutoTestCon, IEEE, 1-5. doi:10.1109/AUTEST.2016.7589570

Douglas-Smith, D., Iwanaga, T., Croke, B. F. W., & Jakeman, A. J. (2020). Certain trends in uncertainty and sensitivity analysis: An overview of software tools and techniques. *Environmental Modelling and Software*, *124*, 1-19. doi:10.1016/j.envsoft.2019.104588

Drabble, L., Trocki, K. F., Salcedo, B., Walker, P. C., & Korcha, R. A. (2015). Conducting qualitative interviews by telephone: Lessons learned from a study of alcohol use among sexual minority and heterosexual women. *Qualitative Social Work,15*, 118-133. doi:10.1177/1473325015585613

Dukes, S., Tourtillott, B., Bryant, D., Carter, K., McNair, S., Maupin, G., & Tamminga, C. (2015). Finishing what was started: An analysis of theater research conducted from 2010-2012. *Military Medicine*, *180*(3), 8-13. doi:10.7205/MILMED-D-14-00393

Duran, K., Burns, G., & Snell, P. (2013). Lehman's laws in agile and non-agile projects. *2013 20th Working Conference on Reverse Engineering, IEEE*. doi:10.1109/WCRE.2013.6671304

Dyk, J. V., & Meghzifene, A. (2017). Radiation oncology quality and safety consideration in low-resource settings: A medical physics perspective. *Seminars in Radiation Oncology*, *27*(2), 124-135. doi:10.1016/j.semradonc.2016.11.004

Eckhart, M., Meixner, K., Winkler, D., & Ekelhart, A. (2019). Securing the testing process for industrial automation software. *Computer & Security*, *85*, 156-180. doi:10.1016/j.cose.2019.04.016

Elberzhager, F., Munch, J., & Assman, D. (2014). Analyzing the relationships between inspections and testing to provide a software testing focus. *Information and Software Technology*, *56*(7), 793-806. doi:10.1016/j.infsof.2014.02.007

Eler, M. M., Endo, A. T., & Durelli, V. H. S. (2016). An empirical study to quantify the characteristics of Java programs that may influence symbolic execution from a unit testing perspective. *Journal of Systems and Software*, *121*, 281-297. doi:10.1016/j.jss.2016.03.020

Em, A. R., & Reedy, E. M. (2015). Software test automation: An algorithm for solving system management automation problems. *Procedia Computer Science*, *46*, 949-956. doi:10.1016/j.procs.2015.01.004

Emam, S. S., & Miller, J. (2015). Test case prioritization using extended diagraphs. *ACM Transactions on Software Engineering and Methodology*, *25*, 6:1-6:41. doi:10.1145/2789209

Engstrom, E., & Petersen, K. (2015). Mapping software testing practice with software testing research-serp-test taxonomy. *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE*, 2015, 1-4. doi:10.1109/ICSTW.2015.7107470

Estrada, L. M., & Koolen, M. (2018). Audiovisual media annotation using qualitative data analysis software: A comparative analysis. *The Qualitative Report*, *23*(13), 40-60. http://nsuworks.nova.edu/tqr

Evans, S., Taber, W., Drain, T., Smith, J., Wu, H. C., Guevara, M., … & Evans, J. (2018). Monte: The next generation of mission design and navigation software, *CEAS Space Journal*, *10*, 79-86. doi:10.1007/s12567-017-0171-7

Fadel, E., Gungor, V. C., Nassef, L., Akkari, N., Abbas-Malik, M. G., Almasri, S., & Akyildiz, I. F. (2015). A survey on wireless sensor networks for smart grid. *Computer Communications*, *71*, 22-23. doi:10.1016/j.comcom.2015.09.006

Feddock, S. (2016). An analysis of the software selection process using waterfall versus agile methodologies: A simulation study. (Doctoral dissertation, Pace University). Proquest id:10128878

Fernandez, D. M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetro, A., … Wieringa, R. (2017). Naming the pain in requirements engineering. *Empirical Software Engineering*, *22*(5), 2298-2338. doi:10.1007/s10664-016-9451-7

Ferrell, O. C., & Ferrell, L. (2016). Ethics and social responsibility in marketing channels and supply chains: An overview. *Journal of Marketing Channels*, *23*, 2-10. doi:10.1080/1046669X.2016.1147339

Fischer-Lokou, J., Gueguen, N., Lamy, L., Martin, A., & Bullock, A. (2014). Imitation on mediation: Effects of the duration of mimicry on reaching agreement. *Social Behavior and Personality, 42*(2), 189-195. doi:10.2224/sbp.2014.42.2.189

Fitzgerald, B., & Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176-189. doi:10.1016/j.jss.2015.06.063

Forster, D. G., & Borasky, D. (2018). Adults lacking capacity to give consent when is it acceptable to include them in research. *Therapeutic Innovation & Regulatory Science, 52*(3), 275-279. doi:10.1177/2168479018770658

Fortineau, V., Paviot, T., & Lamouri, S. (2019). Automated business rules and requirements to enrich product-centric information. *Computers in Industry, 104,* 22-33. doi:10.1016/j.compind.2018.10.001

Fusch, P., Fusch, G. E., & Ness, L. R. (2018). Denzin's paradigm shift: Revisiting triangulation in qualitative research. *Journal of Social Change, 10,* 19-32. doi:10.5590/JOSC.2018.10.1.02

Fusch, P. I., & Ness, L. R. (2015). Are we there yet? Data saturation in qualitative research. *The Qualitative Report*, *20*, 1408–1416. Retrieved from https://nsuworks.nova.edu/tqr

Fylaktopoulos, G., Skolarikis, M., Popadopoulos, I., Goumas, G., Sotiropoulos, A., & Maglogiannis, I. (2018). A distributed modular platform for the development of cloud-based applications. *Future Generation Computer Systems*, *78,* 127-141. doi:10.1016/j.future.2017.02.035

Gario, A., Andrews, A., & Hagerman, S. (2015). Fail-safe testing of safety-critical systems: A case study and efficiency analysis. *Software Quality Journal*, *26*, 3-48. doi:10.1007/s11219-015-9283-5

Garousi, G., Garousi-Yusifoglu, V., Ruhe, G., Zhi, J., Moussavi, M., & Smith, B. (2015). Usage and usefulness of technical software documentation: An individual case study. *Information and Software Technology*, *57*, 664-682. doi:10.1016/j.infsof.2014.08.003

Garousi, V., Felderer, M., & Hacaloğlu, T. (2017). Software test maturity assessment and test process improvement: A multivocal literature review. *Information and Software Technology*, *85*, 16-42. doi:10.1016/j.infsof.2017.01.001

Garousi, V., Felderer, M., Karapicak, C. M., & Yilmaz, U. (2018a). Testing embedded software: A survey of the literature. *Information and Software Technology*, *104*, 14-45. doi:10.1016/j.infsof.2018.06.016

Garousi, V., Felderer, M., Karapicak, C. M., & Yilmaz, U. (2018b). What we know about testing embedded software. *IEEE Software*, *35*(4), 62-69. doi:10.1109/MS.2018.2801541

Garousi, V., & Kucuk, B. (2018). Smells in software test code: A survey of knowledge in industry and academia. *Journal of Systems and Software*, *138*, 52-81. doi:10.1016/j.jss.2017.12.013

Garousi, V., & Mantyla, M. (2016a). A systematic literature review of literature reviews in software testing. *Information and Software Technology*, *80*, 195-216. doi:10.1016/j.infsof.2016.09.002

Garousi, V., & Mantyla, M. K. (2016b). When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology*, *76*, 92-117. doi:10.1016/j.infsof.2016.04.015

Garousi, V., Ozkan, R., & Betin-Can, A. (2018). Multi-objective regression test selection in practice: An empirical study in the defense software industry. *Information & Software Technology*, *103*, 40-54. doi:10.1016/j.infsof.2018.06.007

Garousi, V., & Pfahl, D. (2016). When to automate software testing? A decision-support approach based on process simulation. *Journal of Software Evolution and Process*, *28*, 272-285. doi:10.1002/smr.1758

Gelinas, L., Wertheimeir, A., & Miller, F. G. (2016). When and why is research without consent permissible? *Hastings Center Report*, *46*(2), 35-43. doi:10.1002/hast.548

Gentles, S., Charles, C., Ploeg, J., & McKibbon, K. A. (2015). Sampling in qualitative research: Insights from an overview of the methods literature. *The Qualitative Report*, *20*(11), 1772-1789. Retrieved from Proquest Central Databases.

Gibbins, J., Bhatia, R., Forbes, K., & Reid, C. M. (2014). What do patients with advanced incurable cancer want from the management of their pain? A qualitative study. *Palliative Medicine*, *28*, 71-78. doi:10.1177/0269216313486310

Glaser, B., & Strauss, A. (1967). The discovery of grounded theory: Strategies for qualitative research. Chicago: Aldine

Glasgow, R. E., Huebschmann, A. G., & Brownson, R. C. (2018). Expanding the consort figure: Increasing transparency in reporting on external validity. *American*

Journal of Preventive Medicine, *55*(3), 422-430. doi:10.1016/j.amepre.2018.04.044

Godboley, S., Panda, S., Dutta, A., & Mohapatra, D. P. (2017). An automated analysis of the branch coverage and energy consumption using concolic testing. *Arabian Journal for Science and Engineering*, *42*(2), 619-637. doi:10.1007/s13369-016-2284-2

Godfrey, M. W., & German, D. M. (2014). On the evolution of Lehman's laws. *Journal of Software: Evolution and Process*, *26*(7), 613-619. doi:10.1145/2543581.2543595

Goel, J. N., & Mehtre, B. M. (2015). Vulnerability assessment and penetration testing as a cyber defense technology. *Procedia Computer Science*, *57*, 710-715. doi:10.1016/j.procs.2015.07.458

Gojare, S., Joshi, R., & Gaigaware, D. (2015). Analysis and design of selenium web driver automation testing framework. *Procedia Computer Science*, *50,* 341-346. doi:10.1016/j.procs.2015.04.038

Goltz, U., Reussner, R. H., Goedicke, M., Hasselbring, W., Martin, L., & Heuser, B. V. (2015). Design for future: managed software evolution. *Computer Science-Research and Development, 30*(1), 321-331. doi:10.1007/s00450-014-0273-9

Gomez, O. S., Cortés-Verdín, K., & Pardo, C. J. (2017). Efficiency of software testing techniques: A controlled experiment replication and network meta-analysis. *e-Informatica*, *11*, 77-102. doi:10.5277/e-Inf170104

Goodell, L., Stage, V., & Cooke, N. (2016). Practical qualitative research strategies: Training interviewers and coders. *Journal of Nutrition Education and Behavior, 48*(6), 578-585. doi:10.1016/j.jneb.2016.06.001

Goodman, M. W. (2019). A python library for deep linguistic resources. 2019 Pacific Neighborhood Consortium Annual Conference & Joint Meetings, IEEE, 1-7. Retrieved from http://www.ieeeplore.ieee.org

Granli, W., Burchell, J., Hammouda, I., & Knauss, E. (2015, August). The driving forces of API evolution. In *Proceedings of the 14th International Workshop on Principles of Software Evolution, ACM.* doi:10.1145/2804360.2804364

Grieb, S. D., Eder, M., Smith, K. C., Calhoun, K., & Tandon, D. (2015). Qualitative research and community-based participatory research: Considerations for effective dissemination in the peer-reviewed literature. *Progress in Community Health Partnerships, 9*, 275-282. doi:10.1353/cpr.2015.0041

Groce, A., Alipour, M. A., Chaoqiang, Z., Yang, C., & Regehr, J. (2016). Cause reduction: Delta debugging, even without bugs. *Software Testing, Verification and Reliability, 26*, 40-68. doi:10.1002/stvr.1574

Grubb, P., & Takang, A. A. (2003). *Software maintenance: concepts and practice*. River Edge, NJ: World Scientific.

Guan, F., Peng, L., Perneel, L., & Timmerman, M. (2016). Open source freertos as a case study in realtime operating system evolution. *The Journal of Systems and Software*, *118*, 19-35. doi:10.1016/j.jss.2016.04.063

Gupta, H. (2018). Evaluating service quality of airline industry using hybrid best worst method and vikor. *Journal of Air Transport Management*, *68*, 35-47. doi:10.1016/j.jairtraman.2017.06.001

Gupta, P., Mehlawat, M. K., & Mahajan, D. (2019). Multiobjective optimization framework for software maintenance, component evaluation and selection involving outsourcing, redundancy and customer to customer relationship. *Information Sciences*, *483*, 21-52. doi:10.1016/j.ins.2019.01.017

Gupta, S., & Singh, P. (2017). Comprehending scenario-level software evolution using calling context trees. *2017 International Conference on Information Technology*, *IEEE*, 125-130. doi:10.1109/ICIT.2017.33

Haahr, A., Norlyk, A., & Hall, E. O. (2014). Ethical challenges embedded in qualitative research interviews with close relatives. *Nursing Ethics*, *21*, 6-15. doi:10.1177/0969733013486370

Haegele, J. A., & Hodge, S. R. (2015). Quantitative methodology: A guide for emerging physical education and adapted physical education researchers. *Physical Educator*, *72*(5), 59-75. doi:10.18666/TPE-2015-V72-I5-6133

Haitzer, T., Navarro, E., & Zdun, U. (2017). Reconciling software architecture and source code in support of software evolution. *The Journal of Systems and Software*, *123*, 119-144. doi:10.1016/j.jss.2016.10.012

Halcomb, E., & Hickman, L. (2015). Mixed methods research. *Nursing Standard (2014+)*, *29*(32), 41-47. doi:10.7748/ns.29.32.41.e8858

Hamad, R. M. H. (2018). Automation testing and monitoring lab on the cloud for IoT smart fleet system. *Proceedings of the Fourth International Conference on Engineering & MIS, ACM*, *42*, 1-7. doi:10.1145/3234698.3234740

Hamill, M., & Goseva-Popstojanova, K. (2017). Analyzing and predicting effort associated with finding software faults. *Information and Software Technology*, *87,* 1-18. doi:10.1016/j.infsof.2017.01.002

Hamlet, D. (2015). Theory of software testing with persistent state. *IEEE Transactions on Reliability*, *64*(3), 1098-1115. doi:10.1109/TR.2015.2436443

Hammer, M. J. (2016). Informed consent in the changing landscape of research. *Oncology Nursing Forum*, *43*(5), 558-5610. doi:10.1188/16.ONF.558-560

Hanna, M., Aboutabl, A. E., & Mostafa, M. S. M. (2018). Automated software testing framework for web applications. *International Journal of Applied Engineering Research*, *13*(11), 9758-9767. Retrieved from http://www.ripublication.com

Hashem, I. A. T., Yaqoob, I., Annuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of big data on cloud computing: Review and open research issues. *Information Systems*, *47*, 98-115. doi:10.1016/j.is.2014.07.006

Hatton, L., Spinellis, D., & van Genuchten, M. (2017). The long-term growth rate of evolving software: Empirical results and implications. *Journal of Software: Evolution and Process*, *29*(5), 1847. doi:10.1002/smr.1847

Heeager, L. T., & Nielsen, P. A. (2018). A conceptual model of agile software development in a safety-critical context: A systematic literature review.

Information and Software Technology, 103, 22-39. doi:10.1016/j.infsof.2018.06.004

Heeager, L. T., & Rose, J. (2015). Optimising agile development practices for the maintenance operation: Nine heuristics. *Empirical Software Engineering, 20*(6), 1762-1784. doi:10.1007/s10664-014-9335-7

Henard, C., Papadakis, M., Harman, M., Jia, Y., & LeTraon, Y. (2016). Comparing white box and black box test prioritization. *ICSE '16 Proceedings of the 38th International Conference on Software Engineering, ACM*, 523-534. doi:10.1145/2884781.2884791

Henningsen, M. J., Sort, R., Møller, A. M., & Herling, S. F. (2018). Peripheral nerve block in ankle fracture surgery: A qualitative study of patients' experiences. *Anaesthesia* (Oxford), *73*, 49-58. doi:10.1111/anae.14088

Herraiz, I., Rodriguez, D., Robles, G., & Gonzalez-Barahona, J. M. (2013). The evolution of the laws of software evolution. *ACM Computing Surveys*, *46*(2), 1-32. doi:10.1145/2543581.2543595

Heuser, B. V., Fay, A., Schaefer, I., & Tichy, M. (2015). Evolution of software in automated production systems: Challenges and research directions. *The Journal of Systems & Software*, *110*, 54-84. doi:10.1016/j.jss.2015.08.026

Hinsen, K. (2019). Dealing with software collapse. *Computing in Science and Engineering*, IEEE. *21*(3), 104-108. doi:10.1109/MCSE.2019.2900945

Holloway, I., & Galvin, K. (2016). *Qualitative research in nursing and healthcare*. (4th ed.) John Wiley & Sons, UK: Blackwell Publishing.

Hooda, I., & Chhillar, R. S. (2015). Software test process, testing types and techniques. *International Journal of Computer Applications*, *111*(13), 10-14. Retrieved from http://www.ijcaonline.org

Hora, A., Silva, D., Valente, M. T., & Robbes, R. (2018). Assessing the threat of untracked changes in software evolution. *Proceedings of the 40th International Conference on Software Engineering, ACM*. 1102-1113. doi:10.1145/3180155.3180212

Horga, G., Kaur, T., & Peterson, B. (2014). Annual research review: Current limitations and future directions in MRI studies of child-and adult-onset developmental psychopathologies. *Journal of Child Psychology and Psychiatry*, *55*, 659-680. doi:10.1111/jcpp.12185

Hou, R., Zhou, H., Hu, K., Din, Y., Yang, X., Xu, G., … & Ma, Y. (2016). Thorough documentation of the accidental aspiration and ingestion of foreign objects during dental procedure is necessary: Review and analysis of 617 cases. *Head and Face Medicine*, *12*(23), 1-8. doi:10.1186/s13005-016-0120-2

Hoyland, S., Hollund, J. G., & Olsen, O. E. (2015). Gaining access to a research site and participants in medical and nursing research: A synthesis of accounts. *Medical Education, 49*(2), 224-232. doi:10.1111/medu.12622

Huang, J. (2017). Application of kano model in requirements analysis of y company's consulting project. *American Journal of Industrial and Business Management*, *7,* 910-918. doi:10.4236/ajibm.2017.77064

Huang, J. C. (2014). Don’t fire the architect! Where were the requirements. *IEEE Software*, *31*(2), 27-29. doi:10.1109/MS.2014.34

Huang, J. C., & Wu, T. J. (2018). Development trend of quantity of patents of china patent law. *Journal of Discrete Mathematical Sciences & Cryptography*, *21*(2), 399-403. doi:10.1080/09720529.2018.1449320

Huber, S. T., Kuhm, T., & Sachse, C. (2018). Automated tracing of helical assemblies form electron cryo-micrographs. *Journal of Structural Biology*, *202,* 1-12. doi:10.1016/j.jsb.2017.11.013

Huckabee, W. A. (2015). Requirements engineering in an agile software development environment. *Defense Acquisition Research Journal*, *22*, 394-415. Retrieved from http://www.dau.mil/publications/DefenseARJ/default.aspx

Hussain, A., Razak, H. A., & Mkpojiogu, E. O. C. (2017). The perceived usability of automated testing tools for mobile applications. *Journal of Engineering Science and Technology*, *12*(4), 89-97. Retrieved from http://penerbit.uthm.edu.my/ojs/index.php/JST/index

Huzoree, G., & Ramdoo, V. D. (2015). A systematic study on requirement engineering processes and practices in mauritus. *International Journal of Advanced Research in Computer Science and Software Engineering*, *5*(2), 40-46. Retrieved from www.ijarcsse.com

IEEE (2013). IEEE international standard-software & systems engineering-software testing-part 3: Test documentation. *IEEE*. 1-138. doi:10.1109/IEEESTD.2013.6588540

IEEE (2017). IEEE standard for system, software, and hardware verification and validation. *IEEE*. doi:10.1109/IEEESTD.2017.8055462

IEEE (2018). IEEE 829-2008 ieee standard for software & system test documentation. Retrieved from https://standards.ieee.org/standard

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., & Shamshirband, S. (2015). A systematic literature review on agile requirements, engineering practices and challenges. *Computers in Human Behavior*, *51*, 915-929. doi:10.1016/j.chb.2014.10.046

Iniesto, F., McAndrew, P., Minocha, S., & Coughlan, T. (2016). Accessibility of moocs: Understanding the provider perspective. *Journal of Interactive Media in Education*, *20*, 1-10. doi:10.5334/jime.430

International Software Testing Qualifications Board (2018). Foundation level syllabus. Retrieved from http://www.istqb.org

Itkonen, J., Mantyla, M. V., & Lassenius, C. (2016). Test better by exploring: Harnessing human skills and knowledge. *IEEE Software*, IEEE, *33*(4), 90-96. doi:10.1109/MS.2015.85

Ivanov, V., Reznik, A., & Succi, G. (2018). Comparing the reliability of software systems: A case study on mobile operating systems. *Information Sciences*, *423*, 398-411. doi:10.1016/j.ins.2017.08.079

Jacob, P. M., & Prasanna, M. (2016). A comparative analysis on black-box testing strategies. *2016 International Conference on Information Science (ICIS)*, IEEE, 1-6, doi:10.1109/INFOSCI.2016.7845290

Jamil, M. A., Arif, M., Abubakar, N. S. A., & Ahmad, A. (2016). Software testing techniques: A literature review. *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M), IEEE*. doi:10.1109/ICT4M.2016.045

Jan, S. R., Shah, S. T. U., Johar, Z. U., Shah, Y., & Khan, F. (2016). An innovative approach to investigate various software testing techniques and strategies. *International Journal of Scientific Research Engineering & Technology*, *2*(2), 682-689. Retrieved from http://www.ijsrset.com

Javed, Y., & Alenezi, M. (2016). Defectiveness evolution in open source software systems. *Procedia Computer Science*, *82*, 107-114. doi:10.1016/j.procs.2016.04.015

Jayaram, R., & Krishnan, R. (2018). Approaches to fault localization in combinatorial testing: a survey. *Smart Computing & Informatics*, *78*, 533-540. doi:10.1007/978-981-105547-8_55

Jiang, Z. M. J. (2015). Load testing large-scale software systems. *IEEE*, *2*(1), 955-956. doi:10.1109/ICSE.2015.304

Joann, S. (2015). Software architecture for developers. *IEEE Software*, *32*(5), 93-96. doi:10.1109/ms.2015.125

Joppen, R., Enzberg, S., Kuhn, A., & Dumitrescu, R. (2019). Data map-method for specification of data flows within production. *Procedia CIRP*, *79*, 461-465. doi:10.1016/j.procir.2019.02.127

Julia, S., Vale, L., & Passos, L. (2016). Functional testing using object workflow nets. *Computing & Informatics*, *35*, 719-743. Retrieved from http://www.cai.sk/ojs/index.php/cai/index

Kaefer, F., Roper, J., & Sinha, P. (2015). A software-assisted qualitative content analysis of news articles: Example and reflections. *Forum Qualitative Sozialforschung*, *16*(2), 1-20. doi:10.17169/fqs-16.2.2123

Kalbandi, I., Pawar, M. V., Nikhilkumar, B. S., & Bachate, R. (2015). Iptv software process and workflow. *Procedia Computer Science*, *50*, 128-34. doi:10.1016/j.procs.2015.04.074

Kandl, S., & Chandrashekar, S. (2015). Reasonability of mcidc for safety-relevant software implemented in programming languages with short circuit elevation. *Computing*, *97*(3), 261-279. doi:10.1007/s00607-014-0418-5

Karpowicz, M. P., Arabas, P., & Niewiadomska-Szynkiewicz, E. (2018). Energy-aware multilevel control system for a network of linux software routers: design and implementation. *IEEE Systems Journal*, *12*, 571-582. doi:10.1109/JSYST.2015.2489244

Kaur, A., & Kaur, K. (2019). Investigation on test effort estimation of mobile applications: Systematic literature review and survey. *Information and Software Technology*, *110*, 56-77, doi:10.1016/j.infsof.2019.02.003

Kaur, A., & Vig, V. (2016). Mining software repositories for empirical validation of laws of software evolution for Java projects. *International Journal of Computational Systems Engineering*, *2*(3), 155-173. doi:10.1504/IJCSYSE.2016.079003

Kaur, A., & Vig, V. (2017). Validating lehman’s laws of growth and familiarity for open source java databases. *Computer Communication, Networking, and Internet Security*, *5*, 429-436. doi:10.1007/978-981-10-3226-4_43

Kaur, S., & Kaur, N. (2015). Software metrics evaluation: An open source case study. *International Journal of Computer Science and Information Technologies*, *6*(2), 1565-1568. Retrieved from http://www.ijcsit.com

Kaur, T., Ratti, N., & Kaur, P. (2014). Applicability of lehman laws on open source evolution: A case study. *International Journal of Computer Applications*, *93*(18), 40-46. Retrieved from http:// www.ijcaonline.org

Kebir, S., Borne, I., & Meslati, D. (2017). A genetic algorithm-based approach for automated refactoring of component-based software. *Information and Software Technology*, *88*, 17-36. doi:10.1016/j.infsof.2017.03.009

Keutel, M., Michalik, B., & Richter, J. (2014). Towards mindful case study research in is: A critical analysis of the past ten years. *European Journal of Information Systems*, *23*(3), 256-272. doi:10.1057/ejis.2013.26

Khachaturian, A. S., Hayden, K. M., Mielke, M. M., Tang, Y., Lutz, M. W., Gold, M., … & Khachaturian, Z. S. (2018). New thinking about two, part two. Theoretical articles for alzheimer's and dementia. *Alzheimer's and Dementia*, *14*(6), 703-706. doi:10.1016/j.jalz.2018.05.002

Khan, F. (2016). An innovative approach to investigate various software testing techniques and strategies. *International Journal of Scientific Research*

Engineering & Technology, *2*(2), 682-689. Retrieved from http://www.ijsrset.com

Khan, R., & Amjad, M. (2016). Performance testing (load) of web applications based on test case management. *Perspectives in Science*, *8*, 355-357. doi:10.1016/j.pisc.2016.04.073

Khan, S. (2014). Qualitative research method: Phenomenology. *Asian Social Science*, *10*, 298-310. doi:10.5539/ass.v10n21p298

Khari, M., Lumar, P., Burgos, D., & Crespo, R. G. (2017). Optimized test suites for automated testing using different optimization techniques. *Soft Computing*, 1-12. doi:10.1007/s00500-017-2780-7

Khatiwada, S., Tushev, M., & Mahmoud, A. (2018). Just enough semantics: An information theoretic approach for ir-based software bug localization. *Information and Software Technology*, *93*, 45-57. doi:10.1016/j.infsof.2017.08.012

Kim, J. H., Seo, M., & David, P. (2015). Alleviating depression only to become problematic mobile phone users: Can face-to-face communication be the antidote. *Computers in Human Behavior*, *51(Part A)*, 440-447. doi:10.1016/j.chb.2015.05.030

Kirac, M. F., Aktemur, B., & Sozer, H. (2018). Visor: A fast image processing pipeline with scaling and translation invariance for test oracle automation of visual output systems. *The Journal of Systems and Software*, *136*, 266-277. doi:10.1016/j.jss.2017.06.023

Kirner, R., & Haas, W. (2014). Optimization compilation with preservation of structural code coverage metrics to support software testing. *Software Testing, Verification, and Reliability, 24*, 184-218. doi:10.1002/stvr.1485

Kitamura, T., Alegroth, E., & Ramler, R. (2017). Industry-academia collaboration in software testing: An overview of taic part 2017. *2017 IEEE International Conference on Software Testing, Verification, & Validation Workshops, IEEE*, 42-43. doi:10.1109/ICSTW.2017.13

Klein, H. J., Polin, B., & Sutton, K. L. (2015). Specific onboarding practices for the socialization of new employees. *International Journal of Selection and Assessment*, *23*(3), 263-283. doi:10.1111/ijsa.12113

Konnola, K., Suomi, S., Makila, T., Jokela, T., Rantala, V., & Lehtonen, T. (2016). Agile methods in embedded system development: Multiple-case study of three industrial cases. *Journal of Systems and Software*, *118*, 134-150. doi:10.1016/j.jss.2016.05.001

Konnola, K., Suomi, S., Makila, T., Jokela, T., Rantala, V., & Lehtonen, T. (2017). Can embedded space system development benefit from agile practices? *Eurasip Journal on Embedded Systems*, *2017*(3), 1-16. doi:10.1186/s13639-016-0040-z

Kononov, V., & Rusakov, V. A. (2018). On the problems of developing klee based symbolic interpreter of binary files. *Procedia Computer Science*, *145*, 275-281. doi:10.1016/j.procs.2018.11.058

Kos, T., Mernik, M., & Kosar, T. (2016). Test automation of a measurement system using a domain-specific modeling language. *Journal of Systems and Software*, *111*, 74-88. doi:10.1016/j.jss.2015.09.002

Kour, G., & Singh, P. (2016). Using lehman's laws to validate the software evolution of agile projects. In *Computational Techniques in Information and Communication Technologies (ICCTICT), IEEE*, 90-96. doi:10.1109/ICCTIT.2016.7514558

Kramer, J. P., Brandt, J., & Borchers, J. (2016). Using runtime traces to improve documentation and unit test authoring for dynamic languages. *Chi'16 Proceedings of the 2016 Chi Conference on Human Factors in Computing Systems*, ACM, 3232-3237. doi:10.1145/2858036.2858311

Kramer, M. (2018). Best practices in systems development lifecycle: An analyses based on the waterfall model. *Review of Business & Finance Studies*, *9*, 77-84. Retrieved from http://www.theIBFR.com

Kropp, M., Meier, A., & Biddle, R. (2016). Teaching agile collaboration skills in the classroom. *In 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET), IEEE*. doi:10.1109/CSEET.2016.27

Kruth, J. G. (2015). Five qualitative research approaches and their applications in parapsychology 1. *The Journal of Parapsychology*, *79*(2), 219-233. Retrieved from https://www.parapsych.org

Kukreja, S., Singhal, A., & Bansal, A. (2015). A critical survey on test management in IT projects. *International Conference on Computing, Communication, & Automation, IEEE*, 791-796. doi:10.1109/CCAA.2015.7148481

Kukulies, J., Faulk, B., & Schmitt, R. H. (2016). Development of optimized test planning procedures for stabilizing ramp-up processes by means of design science research. *Procedia CIRP*, *51*, 93-98. doi:10.1016/j.procir.2016.05.056

Kula, R. G., Ouni, A., German, D. M., & Inoue, K. (2018). An empirical study on the impact of refactoring activities on evolving client-used apis. *Information and Software Technology*, *93*, 186-199. doi:10.1016/j.infsof.2017.09.007

Kumar, C., & Yadav, D. K. (2017). Software defects estimation using metrics of early phases of software development lifecycle. *International Journal of System Assurance Engineering and Management*, *8*(4), 2109-2117. doi:10.1007/s13198-014-0326-2

Kumar, D., & Mishra, K. K. (2016). The impacts of test automation on software's cost, quality and time to market. *Procedia Computer Science*, *79*, 8-15. doi:10.1016/j.procs.2016.03.003

Lakshmi, D. S. (2014). Reflective practice through journal writing and peer observation: A case study. *Turkish Online Journal of Distance Education*, *15*, 189-204. Retrieved from http://www.tojde.anadolu.edu/tr

Lalitha, R., Latha, B., & Sumathi, G. (2016). A software design technique for developing medical experts through use case analysis. *Biomedical Research*, 2. Retrieved from http://www.biomedres.info

Langham, E., Thorne, H., Browne, M., Donaldson, P., Rose, J., & Rockloff, M. (2016). Understanding gambling related harm: A proposed definition, conceptual framework, and taxonomy of harms. *BMC Public Health*, *16*(80), 1-23. doi:10.1186/s12889-016-2747-0

Larrea, M. L. (2017). Black-box testing technique for information visualization. Sequencing constraints with low-level interactions. *Journal of Computer Science & Technology*, *17*, 37-48. Retrieved from http://jcst.ict.ac.cn/

Lawlor, D. A., Tilling, K., & Smith, G. D. (2016). Triangulation in aetiological epidemiology. *International Journal of Epidemiology*, *45*(6). 1866-1886. doi:10.1093/ije/dyw314

Leedy, P. D., & Ormrod, J. E. (2015). *Practical research: Planning and design* (9th ed.). Upper Saddle River, NJ: Merrill Prentice-Hall.

Lehman, M. M. (1996). Laws of software evolution revisited. *Proceedings of the 5th European workshop on software process technology*, *1149*, 108-124. doi:10.1007/BFb0017737

Lehman, M. M., & Ramil, J. F. (2002). Software evolution and software evolution processes. *Annals of Software Engineering*, *14*(1-4), 275-309. doi:10.1023/A:102055752

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., & Turski, W. M. (1997). Metrics and laws of software evolution-the nineties view. *Proceedings Fourth International Software Metrics Symposium, IEEE.* doi:10.1109/METRIC.1997.637156

Lei, H., Ganj-eizadeh, F., Jayachandran, P. K., & Ozcan, P. (2017). A statistical analysis of the effects of scrum and kanban on software development projects. *Robotics & Computer-Integrated Manufacturing*, *43*, 59-67. doi:10.1016/j.rcim.2015.12.001

Leins, D. A., Fisher, R. P., Pludwinski, L., Rivard, J., & Robertson, B. (2014). Interview protocols to facilitate human intelligence sources' recollections of meetings. *Applied Cognitive Psychology*, *28*(6), 926-935. doi:10.1002/acp.3041

Lemos, O. A. L., Silveira, F. F., Ferrari, F. C., & Garcia, A. (2018). The impact of software testing education on code reliability: An empirical assessment, *The Journal of Systems and Software*, *137*, 497-511. doi:10.1016/j.jss.2017.02.042

Leotta, M., Clerissi, D., Olianas, D., Ricca, F., Ancona, D., Delzanno, G., … & Ribaudo, M. (2018). An acceptance testing approach for internet of things systems. *IET Software*. doi:10.1049/iet-sen.2017.0344

Liebel, G., Tichy, M., Knauss, E., Ljungkrantz, O., & Stielbauer, G. (2018). Organisation and communication problems in automotive requirements engineering. Requirements Engineering, *23*, 145-167. doi:10.1007/s00766-016-0261-7

Lincoln, Y., & Guba, E. (1985). *Naturalistic inquiry.* Newbury Park, CA: Sage.

Liu, L., Eisingerich, A. B., Auh, S., Merlo, O., & Chun, H. E. H. (2015). Service firm performance transparency: How, when, and why does it pay off? *Journal of Service Research*, 1-17, doi:10.1177/1094670515584331

Lonetti, F., & Marchetti, E. (2018). Chapter three-emerging software testing technologies. *Advances in Computers*, *108*, 91-143. doi:10.1016/bs.adcom.2017.11.003

Low, J. K., Crawford, K., Manias, E., & Williams, A. (2016). A compilation of consumers' stories: The development of a video to enhance medication adherence in newly transplanted kidney recipients. *Journal of Advanced Nursing. 72*(4), 813-824. doi:10.1111/jan.12886

Lub, V. (2015). Validity in qualitative evaluation linking purposes, paradigms, and perspectives. *International Journal of Qualitative Methods*, *14*(5), 1-8. doi:10.1177/1609406915621406

Lucas, G. M., Rizzo, A., Gratch, J., Scherer, S., Stratou, G., Boberg, J., & Morency, L. P. (2017). Reporting mental health symptoms: Breaking down barriers to care with virtual human interviewers. *Frontiers in Robotics and AI*, *4*, 51. doi:10.3389/frobt.2017.00051

Lucassen, G., Dalpiaz, F., E. M. van der Werf, J. M., & Brinkkemper, S. (2015). Forging high-quality user stories: towards a discipline for agile requirements. *2015 IEEE 23rd International Requirements Engineering Conference, IEEE*, doi:10.1109/RE.2015.7320415

Luckmann, P. (2015). Towards identifying success factors for cross-cultural project customer engagement: A literature review. *Procedia Computer Science*, *64*, 324-333. doi:10.1016/j.procs.2015.08.496

Luetsch, K., & Rowett, D. (2016). Developing interprofessional communication skills for pharmacists to improve their ability to collaborate with other professions. *Journal of Interprofessional Care, 30*(4), 458-465. doi:10.3109/13561820.2016.1154021

Lui, C., Kim, J., Kumarasiri, A., Mayyas, E., Brown, S. L., Wen, N., … Chetty, I. J. (2018). An automated does tracking system for adaptive radiation therapy. *Computer Methods and Programs in Biomedicine, 154*, 1-8. doi:10.1016/j.cmpb.2017.11.001

Lyu, M. R. T. (2002). Software reliability theory. *Encyclopedia of Software Engineering.* Hoboken, NJ: John Wiley & Sons.

Machado, I. C., McGregor, J. D., Cavalcanti, Y. C., & Almeida, E. S. d. (2014). On strategies for testing software product lines: A systematic literature review. *Information and Software Technology, 56*, 1183-1199. doi:10.1016/j.infsof.2014.04.002

Maher, C., Hadfield, M., Hutchings, M., & de Eyto, A. (2018). Ensuring rigor in qualitative data analysis: A design research approach to coding combining nvivo with traditional material methods. *International Journal of Qualitative Methods*. *17*, 1-13. doi:10.1177/1609406918786362

Maisikeli, S. G. (2016). Evaluation of software degradation and forecasting future development needs in software evolution. *International Journal of Software Engineering, 7*(6), 49-64. doi:10.5121/ijsea.2016.7604

Malik, S. (2017). Software testing: Essential phase of sdlc and a comparative study of software testing techniques. *International Journal of System & Software Engineering, 5*(2), 38-45. Retrieved from http://www.publishingindia.com/ijsse/

Malterud, K., Siersma, V. D., & Guassora, A. D. (2016). Sample size in qualitative interview studies: Guided by information power. *Qualitative Health Research, 26*(13), 1753-1760. doi:10.1177/1049732315617444

Manikumar, T., Keumar, J. S. K., & Maruthamuth, R. (2016). Automated test data generation for branch testing using incremental genetic algorithm. *Sadhana*, *41*(9), 959-976. doi:10.1007/s12046-016-0536-1

Mariani, L., Pezze, M., & Zuddas, D. (2015). Chapter four-recent advances in automatic black box-testing. *Advances in Computers*, *99*, 157-193. doi:10.1016/bs.adcom.2015.04.002

Marshall, C., & Rossman, G. B. (2016). *Designing qualitative research* (6th ed). Thousand Oaks, CA: Sage.

Martin, J. (2017). Agile organizational change leveraging learnings from software development. *OD Practitioner*, *49*(3), 39-41. Retrieved from https://www.odnetwork.org/page/ODPractitioner

Martin, J. W. (2016). Collecting and processing crustaceans: An introduction. *Journal of Crustacean Biology*, *36*(3), 393-395. doi:10.1163/1937240X-00002436

Mashia, E. O., van Wyk, N. C., & Leech, R. (2019). Support of adolescents to resist peer pressure and coercion to sexual activity. *International Nursing Review*, *66*(3), 416-424. doi:10.1111/inr.12512

Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, P. (2015). Empirical study of agile software development methodologies: A comparative analysis. *ACM SIGSOFT Software Engineering Notes*, *40*, 1-6. doi:10.1145/2693208.2693233

Mattman, I., Gramlich, S., & Kloberdanz, H. (2015). The inscrutable jungle of quality criteria-how to formulate requirements for a successful product development. *Procedia CIRP*, *36*, 153-158. doi:10.1016/j.procir.2015.01.046

McCusker, K., & Gunaydin, S. (2015). Research using qualitative, quantitative or mixed methods and choice based on the research. *Perfusion, 30*(7), 537-542. doi:10.1177/0267659114559116

Meiliana, D., Septian, I., & Alianto, R. S. (2018) Comparison analysis of android gui testing frameworks by using an experimental study. *Procedia Computer Science, 135*, 736-748, doi:10.1016/j.procs.2018.08.211

Memon, M. S., Lee, Y. H., & Mari, S. I. (2015). Group multi-criteria supplier selection using combined grey systems theory and uncertainty theory. *Expert Systems Applications*, *42*, 7951-7959. doi:10.1016/j.eswa.2015.06.018

Mergel, I. (2016). Agile innovation management in government: A research agenda. *Government Information Quarterly*, *33,* 516-523. doi:10.1016/j.giq.2016.07.004

Merriam, S. B. (2014). *Qualitative research: A guide to design and implementation*. San Francisco, CA: John Wiley & Sons

Milajic, A., Beljakovic, D., Davidovic, N., Vatin, N., & Murgul, V. (2015). Using the big-bang-big crunch algorithm for rational design of an energy-plus building, *Procedia Engineering*, *117*, 911-918. doi:10.1016/j.proeng.2015.08.178

Misra, S. C., Bisui, S., & Mahapatra, G. (2018). Trust issues in ERP implementation: modeling and analysis. *Software Quality Professional,* 20(3), 4-16. Retrieved from http://www.asq.org

Mohammed, N. M., Niazi, M., Alshayeb, M., & Mahmood, S. (2017). Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces*, *50*, 107-115. doi:10.1016/j.csi.2016.10.001

Mohan, M., & Shrimali, T. (2017). Hybrid data approach for selecting effective test cases during the regression testing. *International Journal on Smart Sensing & Intelligent Systems*, *10*, 1-24. Retrieved from http://s2is.org/

Moon, K., Brewer, T., Januchowski-Hartley, S., & Blackman, D. (2016). A guideline to improve qualitative social science publishing in ecology and conservation journals. *Ecology and Society*, *21*(3), 17-38. doi:10.5751/ES-08663-210317

Moran, A. (2015). Agile project management. In Managing Agile. 71-101. Springer, Cham.

Morgan, S. J., Pullon, S. R., Macdonald, L. M., McKinlay, E. M., & Gray, B. V. (2017). Case study observational research: A framework for conducting case study research where observation data are the focus. *Qualitative Health Research*, *27*(7), 1060-1068. doi:10.1177/1049732316649160

Morrison, A. D., & Luttenegger, K. C. (2015). Measuring pedagogical content knowledge using multiple points of data. *The Qualitative Report*, *20*(6), 804-816. Retrieved from http://nsuworks.nova.edu/tqr/

Morse, J. M. (2015). Critical analysis of strategies for determining rigor in qualitative inquiry. *Qualitative Health Research*, *25*(9), 1212-1222. doi:10.1177/1049732315588501

Muller, R., Vette, M., & Horauf, L. (2015). An adaptive and automated bolt tensioning system for the pitch bearing assembly of wind turbines. *Robotics and Computer Integrated Manufacturing, 36,* 119-126. doi:10.1016/j.rcim.2014.12.008

Munir, H., Runeson, P., & Wnuk, K. (2018). A theory of openness for software engineering tools in software organizations. *Information and Software Technology*, *97*, 26-45. doi:10.1016/j.infsof.2017.12.008

Murphy, D., & Wright, L. T. (2018). Silver bullet or millstone? A review of success factors for implementation of marketing automation. *Cogent Business & Management*, *5*, 1-10. doi:10.1080/23311975.2018.1546416

Murtazina, M. S., & Avdeenko, T. V. (2019). An ontology-based approach to support for requirements traceability in agile development. *Procedia Computer Science*, *150*, 628-635. doi:10.1016/j.procs.2019.02.044

Nakash, O., Nager, M., & Maymon, Y. K. (2015). What should we talk about? The association between the information exchanged during the mental health intake and quality of the working alliance. *Journal of Counseling Psychology*, *62*(3), 514-520. doi:10.1037/Cou0000074

National Commission for the Protection of Human Subjects of Biomedical and Behavioral Research (1978). The Belmont Report: Ethical principles and guidelines for the protection of human subjects of research. *Bethesda, MD: ERIC clearinghouse*.

Neethidevan, V., & Chandraskaran, G. (2018). Database testing using selenium web driver-a case study. *International Journal of Pure and Applied Mathematics*, *118*(8), 559-566. Retrieved from http://www.ijam.eu

Neuman, D. (2014). Qualitative research in educational communications and technology: A brief introduction to principles and procedures. *Journal of Computing in Higher Education*, *26*, 69-86. doi:10.1007/s12528-014-9078-x

Nichita, D. V. (2018). Volume-based phase stability testing at pressure and temperature specifications. *Fluid Phase Equilibria*, *458*, 123-141. doi:10.1016/j.fluid.2017.10.030

Nidagundi, P., & Novickis, L. (2016). Introduction to lean canvas transformation models and metrics in software testing. *Applied Computer Systems*, *19*, 30-36. doi:10.1515/acss-2016-0004

Nidagundi, P., & Novickis, L. (2017). Introducing lean canvas model adaptation in the scrum software testing. *Procedia Computer Science*, *104*, 97-103. doi:10.1016/j.procs.2017.01.078

Nikiforova, A., & Bicevska, Z. (2018). Application of lean principles to improve business processes: A case study in latvian it company. *Baltic Journal of Modern Computing*, *6*(3), 247-270. doi:10.22364/bjmc.2018.6.3.03

Njie, B., & Asimiran, S. (2014). Case study as a choice in qualitative methodology. *Journal of Research & Method in Education*, *4*(3), 35-40. Retrieved from http://www.iosrjournals.org

Nouacer, R., Djemal, M., Niar, S., Mouchard, G., Rapin, N., Gallois, J. P., … & Mac-Eachen, B. (2016). Equitas: A tool-chain for functional safety and reliability improvement in automotive systems. *Microprocessors and Microsystems, 47*, 252-261. doi:10.1016/j.micpro.2016.07.020

Novais, R., Santos, J. A., & Mendonca, M. (2017). Experimentally assessing the combination of multiple visualization strategies for software evolution analysis. *The Journal of Systems and Software, 128*, 56-71. doi:10.1016/j.jss.2017.03.006

Nunes, C. A., Alvarenga, V. O., De Souza Sant'Ana, A., Santos, J. S., & Granato, D. (2015). The use of statistical software in food science and technology: Advantages, limitations, and misuses. *Food Research International*, *75*, 270-280. doi:10.1016/j.foodres.2015.06.011

O'Cathain, A., Goode, J., Drabble, S. J., Thomas, K. J., Rudolph, A., & Hewison, J. (2014). Getting added value from using qualitative research with randomized controlled trials: A qualitative interview study. *Trials*, *15*, 1-20. doi:10.1186/1745-6215-15-215

Odzaly, E. E., Greer, D., & Stewart, D. (2018). Agile risk management using software agents. *Journal of Ambient Intelligence and Humanized Computing*, *9*(3), 823-841. doi:10.1007/s12652-017-0488-2

Ogbodo, I. (2014). Effects of conflicts between developers, testers, and business analysts on software development (Doctoral dissertation, Walden University). Proquest id: 3620039

Oghenovo, E. (2014). Software dysfunction: Why do software fail? *Journal of Computer and Communications, 2,* 25-35. doi:10.4236/jcc.2014.26004

Ojo, A. I., & Popoola, S. O. (2015). Some correlates of electronic health information management system success in nigerian teaching hospitals. *Biomedical Informatics Insights, 7,* 1-9. doi:10.4137/BII.s20229

Okoye, K., Naeem, U., & Islam, N. (2017). Semantic fuzzy mining: enhancement of process models and event logs analysis from syntactic to conceptual level. *International Journal of Hybrid Intelligent Systems*, *14*, 67-98. doi:10.3233/HIS-170243

Okwu, P. I., & Onyeje, I. N. (2014). Software evolution: Past, present, and future. *American Journal of Engineering Research*, *3*(5), 21-28. Retrieved from http://www.ajer.org

Olatunji, M. A., Oladele, R. O., & Bajeh, A. O. (2017). Empirical study of continuous change of open source system. *International Journal of Computing & ICT Research*, *11*, 31-52. Retrieved from http://www.ijcir.mak.ac.ug

Oliveira, R. P., & Almeida, E. S. (2016). Evaluating Lehman's laws of software evolution for software product lines, *IEEE Software*, *33*(3), 90-93. doi:10.1109/MS.2016.78

Oliveria, R. P., Santos, A. R., Almeida, E. S., & Gomes, G. S. (2017). Evaluating lehman’s laws of software evolution within software product lines industrial projects. *The Journal of Systems and Software*, *131*, 347-365. doi:10.1016/j.jss.2016.07.038

O'Sullivan, D., & Conway, P. F. (2016). Underwhelmed and playing it safe: Newly qualified primary teachers’ mentoring and probationary-related experiences during induction. Irish Educational Studies, *35*(4), 1-18. doi:10.1080/03323315.2016.1227720

Panichella, A., & Molina, U. R. (2017). Java unit testing tool competition – Fifth round. *2017 IEEE/ACM 10th International Workshop on Search Based Software Testing, IEEE*, 32-38. doi 10.1109/SBST.2017.7

Papadakis, M., Ali, S., & Perrouin, G. (2019). Editorial to the theme section on model-based testing. *Software and Systems Modeling*, *18*(2), 795-796. doi:10.1007/s10270-018-0699-9

Parampreet, K., & Rajeev, S. (2018). A modeling framework for automotive software design and optimal test path generation. *Journal of Intelligent and Fuzzy Systems*, *34*(3), 1731-1742. doi:10.3233/JIFS-169466

Parsons, D., Susnjak, T., & Lange, M. (2014). Influences on regression testing strategies in agile software development environments. *Software Quality Journal*, *22*(4), 717-739. doi:10.1007/s11219-013-9225-z

Patel, M. R., Shah, K. S., & Shallcross, M. L. (2015). A qualitative study of physician perspectives of cost-related communication and patients' financial burden with managing chronic disease. *BMC Health Services Research*, *15*, 1-7. doi:10.1186/s12913-015-1189-1

Patenaude, A. F., Pelletier, W., & Bingen, K. (2015). Communications, documentation and training standards in pediatric psychosocial oncology. *Pediatric Blood & Cancer*, *62*, 870-875. doi:10.1002/pbc.25725

Pauly, D., Michalik, B., & Basten, D. (2015). Do daily scrums have to take place each day? A case study of customized scrum principles at an e-commerce company. *2015 48th Hawaii International Conference on Systems Sciences*, *IEEE,* 5074-5083. doi:10.1109/HICSS.2015.601

Pawlak, M., & Poniszewska-Maranda, A. (2018). Software test management approach for agile environments. *Information Systems in Management, 7*, 47-58. doi:10.22630/ISIM.2018.7.1.5

Percy, W. H., Kostere, K., & Kostere, S. (2015). Generic qualitative research in psychology. *The Qualitative Report*, *20*(2), 76-85. Retrieved from www.nsuworks.nova.edu/tqr/

Pereira, J. C., & de F. S. M. Russo, R. (2018). Design thinking integrated in agile software development: A systematic literature review. *Procedia Computer Science, 138*, 775-782. doi:10.1016/j.procs.2018.10.101

Peticca-Harris, A., de Gama, N., & Elias, S. (2016). A dynamic process model for finding informants and gaining access in qualitative research. *Organizational Research Methods*, *19*(3), 376-401. doi:10.1177/1094428116629218

Petunova, O., & Berzisa, S. (2017). Test case review processes in software testing. *Information Technology and Management Science, 20,* 48-53. doi:10.1515/itms-2017-0008

Phillips, D. (2004). *The software project manager's handbook: principles that work at work* (2nd ed). Wiley-IEEE Computer Society Press

Ping, P., Xuan, Z., & Xinyue, M. (2017). Research on security test for application software based on spn. *Procedia Engineering*, *174*, 1140-1147. doi:10.1016/j.proeng.2017.01.267

Politowski, C., Fontoura, L. M., Petrillo, F., & Gueheneuc, Y. G. (2018). Learning from the past: a process recommendation system for video game projects using postmortems experiences. *Information and Software Technology, 100,* 103-118. doi:10.1016/j.infsof.2018.04.003

Post, C. (2017). Preservation practices of new media artists. *Journal of Documentation, 73*(4), 716-732. doi:10.1108/JD-09-2016-0116

Poth, A. (2016). Effectivity and economical aspects for agile quality assurance in large enterprises. *Journal of Software: Evolution and Process*, *28*(11), 1000-1004. doi:10.1002/smr.1823

Prechelt, L., Schmeisky, H., & Zieris, F. (2016). Quality experience: A grounded theory of successful agile projects without dedicated testers. *In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE*. 1017-1027. doi:10.1145/2884781.2884789

Primiero, G., & Raimondi, F. (2015). Software theory change for resilient near-complete specifications. *Procedia Computer Science*, *52*, 988-995. doi:10.1016/j.procs.2015.05.091

Purohit, S. K., & Sharma, A. K. (2016). Evaluation of programming tools for the development of data mining driven forecasting software tool for quality function deployment. *Computational Intelligence & Communication Technology (CICT), IEEE*, 134-139. doi:10.1109/CICT.2016.35

Rabah, S., Belqasmi, F., Mizouni, R., & Dssouli, R. (2016). An elastic hybrid sensing platform: Architecture and research challenges. Procedia *Computer Science*, *94*, 113-120. doi:10.1016/j.procs.2016.08.019

Rais, A. A. (2016). Interface based software testing. *Journal of Systems Integration*, *7*(4), 46. doi:10.20470/jsi.v7i4.277

Ramanathan, S., Faulkner, G., Berry, T., Deshpande, S., Latimer-Cheung, A. E., Rhodes, R. E., … & Tremblay, M. S. (2018). Perceptions of organizational capacity to promote physical activity in canada and participaction's influence five years after its relaunch: A qualitative study. Health Promotion and Chronic Disease Prevention in Canada, *38*(4), 170-178. doi:10.24095/hpcdp.38.4.03

Ramos, F. M., Kreutz, D., & Verissimo, P. (2015). Software-defined networks: On the road to the softwarization of networking. *Cutter IT Journal*, 1-7. Retrieved from https://www.cutter.com/

Ranney, M. L., Meisel, Z. F., Choo, E. K., Garro, A. C., Sasson, C., & Morrow, K. (2015). Interview-based qualitative research in emergency care part II: Data collection, analysis and results reporting. *Academic Emergency Medicine*. *22*(9), 1103-1112. doi:10.1111/acem.12735

Rapport, F., Clement, C., Doel, M. A., & Hutchings, H. A. (2015). Qualitative research and its methods in epilepsy: Contributing to an understanding of patients lived experiences of the disease. *Epilepsy and Behavior, 45*, 94-100. doi:10.1016/jiyebeh.2015.01.040

Raschke, W., Zilli, M., Loineg, J., Weiss, R., Steger, C., & Kreiner, C. (2015). Where does all the waste come from? *Journal of Software: Evolution and Process*, *27*, 584-590. doi:10.1002/smr.1732

Rastogi, V. (2015). Software development life cycle models-comparison, consequences. *International Journal of Computer Science and Information Technologies*, *6*, 168-172. Retrieved from http://www.ijcsit.com

Reeves, D., Howells, K., Panagiotti, M., Bower, P., Sidaway, M., Blakemore, A., & Hann, M. (2018). The cohort multiple random controlled trial design was found to be highly susceptible to low statistical power cord internal validity bias. *Journal of Clinical Epidemiology, 95*, 111-119. doi:10.1016/j.jclinepi.2017.12.008

Rempel, P., & Mader, P. (2015). A quality model for the systematic assessment of requirements traceability. *2015 IEEE 23rd International Requirements Engineering Conference, IEEE.* doi:10.1109/RE.2015.7320420

Rempel, P., & Mader, P. (2017). Preventing defects: The impact of requirements traceability completeness on software quality, *IEEE Transactions on Software Engineering*, *43*(8), 777-797. doi:10.1109/TSE.2016.2622264

Ridder, H. (2017). The theory contribution of case study research designs. *Business Research 10*, 281-305. doi:10.1007/s40685-017-0045-z

Rigoni, A., Manduchi, G., Luchetta, A., Taliercio, C., & Shroder, T. (2018). A framework for the integration of the development process of linux fpga system on chip devices. *Fusion Engineering and Design, 128*, 122-125. doi:10.1016/j.fusengdes.2018.01.042

Robinson, O. C. (2014). Sampling in interview-based qualitative research: A theoretical & practical guide. *Qualitative Research in Psychology*, *11*, 25-41. doi:10.1080/14780887.2013.801543

Rola, P., Kuchta, D., & Kopczyk, D. (2016). Conceptual model of working space for agile (scrum) project team. *Journal of Systems and Software, 118*, 49-63. doi:10.1016/j.jss.2016.04.071

Roldan, M. L., Vegetti, M., Gonnet, S., Leone, H., & Marciszack, M. (2019). An ontology for specifying and tracing requirements engineering artifacts and test artifacts. *CLEI Eletronic Journal*, *22*, 1-19. Retrieved from www.clei.cl/cleiejindex.html

Ross, C., Rogers, C., & Duff, D. (2016). Critical ethnography: an under-used research methodology in neuroscience nursing. *Canadian Journal of Neuroscience Nursing*, *38*, 4-6. Retrieved from http://www.cann.ca

Roy, K., Zvonkovic, A., Goldberg, A., Sharp, E., & Larossa, R. (2015). Sampling richness and qualitative integrity challenges for research with families. *Journal of Marriage and Family*, *77*, 243-260. doi:10.1111/jomf.12147

Ruohonen, J., Hyrynsalmi, S., & Leppanen, V. (2015). Time series trends in software evolution. *Journal of Software: Evolution and Process*, *27*, 990-1015. doi:10.1002/smr.1755

Ryan, J. (2013). Book review: Karin olson, essentials of qualitative interviewing. *Qualitative Research*, *13*(2), 254. doi:10.1177/1468794112450832

Sadath, L., Karim, K., & Gill, S. (2018). Extreme programming implementation in academia for software engineering sustainability. *2018 Advances in Science and Engineering Technology International Conferences (ASET), IEEE*. doi:10.1109/ICASET.2018.8376925

Saeed, S., Khan, F. H., Khan, S. A., & Islam, N. (2018). Conceptions of software testing as a service. *Journal of Fundamental and Applied Sciences,* Retrieved from http://www.jfas.info

Safa, N. S., Maple, C., Furnell, S., Azad, M. A., Perera, C., Dabbagh, M., & Sookhak, M. (2019). Deterrence and prevention-based model to mitigate information security insider threats in organisations. *Future Generation Computer Systems*, *97*, 587-597. doi:10.1016/j.future.2019.03.024

Saini, M., Mehmi, S., & Chahal, K. K. (2016). Understanding open source software evolution using fuzzy data mining algorithm for time series data. *Advances in Fuzzy Systems*, *2016*, 1-13. doi:10.1155/2016/1479692

Salmona, M., Kaczynsk, D., & Smith, T. (2015). Qualitative theory in finance: theory into practice. *Australian Journal of Management*, *40*, 403-413. doi:10.1177/0312896214536204

Sanchez, A. B., Delgado-Perez, P., Segura, S., & Medina-Bulo, I. (2018). Performance mutation testing: Hypothesis and open questions. *Information and Software Technology*, *103*, 159-161. doi:10.1016/j.infsof.2018.06.015

Sanchez-Morcilio, R., & Quiles-Torres, F. (2017). The taxonomy of estimation in software development projects. *Issues in Information Systems*, *18*(3), 116-128. Retrieved from http://www.iacis.org/iis/iis.php

Sapna, P. G., & Balakrishnan, A. (2015). An approach for generating minimal test cases for regression testing. *Procedia Computer Science*, *47*, 188-196. doi:10.1016/j.procs.2015.03.197

Saunders, B., Kitzinger, J., & Kitzinger, C. (2015). Anonymising interview data: challenges and compromise in practice. *Qualitative Research*, *15*(5), 616-632. doi:10.1177/1468794114550439

Scatalon, L. P., Barbosa, E. F., & Garcia, R. E. (2017). Challenges to integrate software testing into introductory programming courses. *2017 IEEE Frontiers in Education Conference, IEEE*, 1-9. doi:10.1109/FIE.2017.8190557

Scheibe, A., Grasso, M., Raymond, H. F., Manyuchi, A., Osmand, T., Lane, T., & Struthers, H. (2018). Modelling the unaids 90-90-90 treatment cascade for gay, bisexual and other men who have sex with men in south africa: Using the findings of a data triangulation process to map a way forward. *AIDS and Behavior*, *22*(3), 853-859. doi:10.1007/s10461-017-1773-y

Schrems, B. M. (2014). Informed consent, vulnerability and the risks of group-specific attribution. *Nursing Ethics*, *21*(7), 829-843. doi:10.1177/0969733013518448

Schroder, M., Raben, H., Kruger, F., Ruscheinski, A., van Rienen, U., Uhrmacher, A., & Spors, S. (2019). Provenance patterns in numerical modelling and finite element simulation processes of bio-electric systems. *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society, IEEE*, 3377-3382. doi:10.1109/EMBC.2019.8856841

Schwartz, A., Puckett, D., Meng, Y., & Gay, G. (2018). Investigating faults missed by test suites achieving high code coverage. *Journal of Systems and Software, 144,* 106-120. doi:10.1016/j.jss.2018.06.024

Seitz, S. (2016). Pixilated partnerships, overcoming obstacles in qualitative interviews via skype: A research note. *Qualitative Research*, *16*(2), 229-235. doi:10.1177/1468794115577011

Sen, S., Marijan, D., & Gotlieb, A. (2018). Certus: An organizational effort towards research-based innovation in software verification and validation. *International Journal of System Assurance Engineering and Management*, *9*(2), 313-322. doi:10.1007/s13198-015-0352-8

Shang, W., Nagappan, M., & Hassan, A. E. (2015). Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Software Engineering*, *20*, 1-27. doi:10.1007/s10664-013-9274-8

Sharma, D., & Chandra, P. (2019). A comparative analysis of soft computing techniques in software fault prediction model development. *International Journal of Information Technology*, *11*, 37-46. doi:10.1007/s41870-018-0211-3

Shehzad, K., & Shaikh, M. U. (2017). A critique base solution on lehman's law. *Science International*, *29*(3), 503-507. Retrieved from http://www.sci-int.com/

Shin, K. W., & Lim, D. J. (2018). Model-based automatic test case generation for automotive embedded software testing. *International Journal of Automotive Technology*, *19*, 107-119. doi:10.1007/s12239-018-0011-6

Shirazi, F., Kazemipoor, H., & Tavakkoli-Moghaddam, R. (2017). Fuzzy decision analysis for project scope management. *Decision Science Letters*, *6*(4), 395-406. doi:10.5267/j.dsl.2017.1.003

Sifeng, L., Tao, L., Xie, N., & Yang, Y. (2016). On the new model system and framework of grey system theory. *Journal of Grey System*, *28*, 1-15. doi:10.1109/GSIS.2015.7301810

Sills, D., Tunks, K., & O'Leary, J. (2017). *Scaling agile for government: using agile on large, complex projects in government*. New York, NY: Deloitte University Press

Singh, A. S. (2014). Conducting case study research in non-profit organisations. *Qualitative Market Research: An International Journal*, *17*, 77–84. doi:10.1108/QMR-04-2013-0024

Singh, S., & Kaur, S. (2017). A systematic literature review: Refactoring for disclosing code smells in object-oriented software. *Ain Shams Engineering Journal*. doi:10.1016/j.asej.2017.03.002

Singhal, N., & Bhola, P. (2017). Ethical practices in community-based research in non-suicidal self-injury: A systematic review. *Asian Journal of Psychiatry, 30*, 127-134. doi:10.1016/j.ajp.2017.08.015

Sipes, J. B. A., Roberts, L. D., & Mullan, B. (2019). Voice-only skype for sue in researching sensitive topics: A research note. *Qualitative Research in Psychology*, 1-17. doi:10.1080/14780887.2019.1577518

Skoulis, I., Vassiliadis, P., & Zarras, A. V. (2015). Growing up with stability: How open-source relational databases evolve. *Information Systems*, *53*, 363-385. doi:10.1016/j.is.2015.03.009

Smada, D., Rotuna, C., Boneca, R., & Petre, I. (2018). Automated code testing system for bug prevention in web-based user interfaces. *Informatica Economica*, *22*(3), 23-32. doi:10.12948/issn14531305/22.3.2018.03

Snyder, E. J., Zhang, W., Jasmin, K. C., Thankachan, S., & Donnelly, L. F. (2018). Gauging potential risk for participants in pediatric radiology by review of over 2,000 incident reports. *Pediatric Radiology*, 1-8. doi:10.1007/s00247-018-4238-1

Sohaib, O., Solanki, H., Dhaliwa, N., Hussain, W., & Asif, M. (2018). Integrating design thinking into extreme programming. *Journal of Ambient Intelligence and Humanized Computing*, 1-8. doi:10.1007/s12652-018-0932-y

Sohn, B. K., Thomas, S., Greenberg, K., & Pollio, H. R. (2017). Hearing the voices of students & teachers: A phenomenological approach to educational research. *Qualitative Research in Education*, *6*(2), 121-148. Retrieved from http://www.hipatiapress.com

Stake, R. (1995). *The art of case study research*. Thousand Oaks, CA: Sage.

Stavova, V., Dedkova, L., Ukrop, M., & Matyas, V. (2018). A large-scale comparative study of beta testers and regular users. *Communications of the ACM*, *61*(2), 64-71. doi:10.1145/3173570

Steinberger, M., Reinhartz-Berger, I., & Tomer, A. (2018). Cross lifecycle variability analysis: Utilizing requirements and testing artifacts. *Journal of Systems and Software*, *143*, 208-230. doi:10.1016/j.jss.2018.04.062

Steinert, Y., & Thomas, A. (2016). When I say… literature reviews. *Medical Education, 50*(4), 398-399. doi:10.1111/medu.12998

Steinke, G. H., Al-Deen, M. S., & LaBrie, R. C. (2017). Innovating information system development methodologies with design thinking. *In Proceedings of the 5th International Conference on Applied Innovations in IT*, 51-55. Retrieved from http://icaiit.org/

Stillwell, P., Hayden, J. A., Rosiers, P. D., Harman, K., French, S. D., & Curran, J. A. (2018). A qualitative study of doctors of chiropractic in a nova scotian practice-based research network: Barriers and facilitators to the screening and management of psychosocial factors for patients with low back pain. *Journal of Manipulative and Physiological Therapeutics*, *4*, 25-33. doi:10.1016/j.jmpt.2017.07.014

Stockman, C. (2015). Achieving a doctorate through mixed methods research. *Electronic Journal of Business Research Methods*, *13*(2), 74-84. Retrieved from http://www.ejbrm.com

Stol, K. J., & Fitzgerald, B. (2015). Theory-oriented software engineering. *Science of Computer Programming*, *101*, 79-98. doi:10.1016/j.scico.2014.11.010

Strandberg, P. E., Enoui, E. P., Afzal, W., Sundmark, D., & Feldt, R. (2019). Information flow in software testing: An interview study with embedded software engineering practitioners. *IEEE Access*, *7*, 46434-46453. doi:10.1109/ACCESS.2019.2909093

Subramanian, G. H., Pendharkar, P. C., & Pai, D. R. (2017). An examination of determinants of software testing and project management effort. *Journal of Computer Information Systems*, *57*(2), 123-129. doi:10.1080/08874417.2016.1183428

Suffian, M. D. M., Fahrurazi, F. R., Ann, L. F., Aman, N. F., & Bajuri, N. (2018). Rating of software trustworthiness via scoring of system testing results. *International Journal of Digital Enterprise Technology*, *1*(1-2), 121-134. doi:10.1504/IJDET.2018.092637

Sun, Q., Wu, J., Rong, W., & Liu, W. (2019). Formative assessment of programming language learning based on peer code review: Implementation and experience report. *Tsinghua Science and Technology*, *24*(4), 423-434. doi:10.26599/TST.2018.9010109

Sun, X., Li, B., Leung, H., Li, B., & Li, Y. (2015). MSR4SM: Using topic models to effectively mining software repositories for software maintenance tasks. *Information and Software Technology*, *66*, 1-12. doi:10.1016/j.infsof.2015.05.003

Sutton, J., & Austin, Z. (2015). Qualitative Research: Data collection, analysis and management. *Canadian Journal of Hospital Pharmacy*, *68*(3), 226-231. https://www.cjhp-online.ca/index.php/cjhp/index

Swarts, J. (2015). Help is in the helping: an evaluation of help documentation in a networked age. *Technical Communication Quarterly*, *24*(2), 164-187. doi:10.1080/10572252.2015.1001298

Tan, T. K., Weerakkody, R., Mrak, M., Ramzan, N., Baroncini, V., Ohm, J. R., … & Sullivan, G. J. (2016). Video quality evaluation methodology and verification testing of hevc compression performance. *IEEE Transactions on Circuits and Systems for Video Technology*, *26*, 76-90. doi:10.1109/TCSVT.2015.2477916

Thomas, D. R. (2017). Feedback from research participants are member-checks useful in qualitative research? *Qualitative Research in Psychology*, *14*, 23-41. doi:10.1080/14780887.2016.1219435

Tissenbaum, M. (2020). I see what you did there! Divergent collaboration and learner transitions from unproductive to productive states in open-ended inquiry. *Computers & Education*, *145*, 1-15. doi:10.1016/j.compedu.2019.103739

Tomar, D., & Agarwal, S. (2016). Prediction of defective software modules using class imbalance learning. *Applied Computational Intelligence and Soft Computing*, *2016*, 6. doi:10.1155/2016/7658207

Tong, A., & Dew, M. A. (2016). Qualitative research in transplantation: Ensuring relevance and rigor. *Transplantation*, *100*(4), 710-712. doi:10.1097/TP.0000000000001117

Tramontana, P., Amalfitano, D., Amatucci, N., & Fasolino, A. R. (2019). Automated functional testing of mobile applications: A systematic mapping study. *Software Quality Journal*, *27*, 149-201. doi:10.1007/s11219-018-9418-6

Tripp, J. F., & Armstrong, D. J. (2018). Agile methodologies: Organizational adoption motives, tailoring, and performance. *Journal of Computer Information Systems*, *58*(2), 170-179. doi:10.1080/08874417.2016.1220240

Trnka, S. (2017). The fifty-minute ethnography: Teaching theory through fieldwork. *The Journal of Effective Teaching*, *17*, 28-34. Retrieved from http://www.uncw.edu/cte/et

Tsunoda, T., Washizaki, H., Fukazawa, Y., Inoue, S., Hanai, Y., & Kanazawa, M. (2018). Empirical study on specification metrics to predict volatility and software defects. *2018 IEEE Region 10 Conference, IEEE*, 2479-2484. doi:10.1109/TENCON.2018.8650274

Tu, H., Lin, Z., & Lee, K. (2019). Automation with intelligence in drug research. *Clinical Therapeutics*, *41*, 2436-2444. doi:10.1016/j.clinthera.2019.09.002

Tuffley, D. (2011). *Software test plans: A how-to guide for project staff.* USA: Altiroa.

Underwood, S. (2016). Exploring organizations' software quality assurance strategies (Doctoral dissertation, Walden University). Proquest id:10162955

Unterkalmsteiner, M., Gorschek, T., Feldt, R., & Klotins, E. (2015). Assessing requirements engineering and software test alignment-five case studies. *The Journal of Systems and Software*, *109*, 62-77. doi:10.1016/j.jss.2015.07.018

U.S. Department of Health & Human Services. (1979). The Belmont Report. Retrieved from http://www.hhs.gov/ohrp/humansubjects/guidance/belmont.html

U.S. Government Accountability Office (GAO) (2014). Healthcare.gov: Ineffective planning and oversight practices underscore the need for improved contract management. Washington, DC: U.S. Government Accountability Office.

Vaismoradi, M., Jones, J., Turunen, H., & Snelgrove, S. (2016). Theme development in qualitative content analysis and thematic analysis. *Journal of Nursing Education and Practice*. *6*(5), 100-110. doi:10.5430/jnep.v6n5p100

Vasanthapriyan, S., Tian, J., Zhao, D., Xiongi, S., & Xiang, J. (2017). An ontology-based knowledge sharing portal for software testing. *IEEE International Conference on Software Quality, Reliability & Security Companion, IEEE*, 472-479. doi:10.1109/QRS-C.2017.82

Versteeg, S., Du, M., Bird, J., Schneider, J. G., Grundy, J., & Han, J. (2016). Enhanced playback of automated service emulation models using entropy analysis. *Proceedings of the International Workshop on Continuous Software Evolution & Delivery, ACM*, 49-55. doi:10.1145/2896941.2896950

Vijayasarathy, L. R., & Butler, C. W. (2016). Choice of software development methodologies. Do organizational project and team characteristics matter? *IEEE Software*, *33*(5). doi:10.1109/MS.2015.26

Vila, E., Novakova, G., & Todorova, D. (2017). Automation testing framework for web applications with selenium webdriver: Opportunities and threats. *2017 Proceedings of the International Conference on Advances in Image Processing, ACM*, 144-150. doi:10.1145/3133264.3133300

Vora, U. (2015). Precepts and evolvability of complex systems. *Procedia Computer Science, 62,* 565-574. doi:10.1016/j.procs.2015.08.533

Vukovic, V., Trninic, J., & Djurkovic, J. (2018). A business software testing process-based model design. *International Journal of Software Engineering and Knowledge Engineering*, *28*(5), 701-749. doi:10.1142/s0218194018500201

Wahler, M., Drofenik, U., & Snipes, W. (2016). Improving code maintainability: A case study on the impact of refactoring. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference*, *IEEE*, 493-501. doi:10.1109/ICSME.2016.52

Wang, H., Wang, G., & Duan, F. (2016). Planning of step-stress accelerated degradation test based on the inverse gaussian process. *Reliability Engineering & System Safety*, *154*, 97-105. doi:10.1016/j.ress.2016.05.018

Wang, Y., Graziotin, D., Kriso, S., & Wagner, S. (2019). Communication channels in safety analysis: An industrial exploratory study. *The Journal of Systems and Software*, *153*, 135-151. doi:10.1016/j.jss.2019.04.004

Wang, Y., Wan, Q., Guo, J., Jin, X., Zhou, W., Feng, X., … Shang, S. (2020). The influence of effective communication, perceived respect and willingness to collaborate on nurses' perceptions of nurse-physician collaboration in China. *Applied Nursing Research*, *41*, 73-79. doi:10.1016/j.apnr.2018.04.005

Wang, Y., Zheng, J., Zhang, A., Zhou, W., & Dong, H. (2018). Visualization maps for the evolution of research hotspots in the field of regional health information networks. *Informatics for Health and Social Care, 43*(2), 186-206. doi:10.1080/17538157.2017.1297304

Weidner, A. K. H., Pauwels, J., McGuire, M., & Davis, A. (2017). Collaboration between acgme and aoa programs to enhance success in single accreditation system: A process paper. *Journal of the American Osteopathic Association, 117*, 705-711. doi:10.7556/jaoa.2017.133

Wohlin, C., Smite, D., & Moe, N. B. (2015). A general theory software engineering: Balancing human, social and organizational capitals. *The Journal of Systems and Software*, *109*, 229-242. doi: 10.1016/j.jss.2015.08.009

Wolgemuth, J. R. (2014). Analyzing for critical resistance in narrative research. *Qualitative Research, 14*(5), 586–602. doi:10.1177/1468794113501685

Wood, L., Burke, E., Byrne, R., Enache, G., & Morrison, A. P. (2016). Semistructured interview measure of stigma (SIMS) in psychosis: Assessment of psychometric properties. *Schizophrenia Research*, 1-6. doi:10.1016/j.schres.2016.06.008

Wright, A., Ash, J. S., Aaron, S., Ai, A., Hickman, T. T., Wiesen, J. F., … & Sittig, D. F. (2018). Best practices for preventing malfunctions in rule-based clinical decision support alerts and reminders: Results of a delphi study. *International Journal of Medical Informatics, 118,* 78-85. doi:10.1016/j.ijmedinf.2018.08.001

Xiao, P., Liu, B., & Wang, S. (2018). Feedback based integrated prediction: Defect prediction based on feedback from software testing process. *The Journal of Systems and Software*, *143*, 159-171. doi:10.1016/j.jss.2018.05.029

Yadav, H. B., & Yadav, D. K. (2015). A fuzzy logic-based approach for phase-wise software defects prediction using software metrics. *Information and Software Technology*, 63, 44-57. doi:10.1016/j.infsof.2015.03.001

Yague, A., Garbajosa, J., Diaz, J., & Gonzalez, E. (2016). An exploratory study in communication in agile global software development. *Computer Standards & Interfaces*, *48*, 184-197. doi:10.1016/j.csi.2016.06.002

Yao, Y., & Liu, J. (2018). Metamorphic testing for oracle problem in integer bug detection. *International Journal of Performability Engineering*, *14*(7), 1481-1486. doi:10.23940/ijpe.18.07.p11.14811486

Yates, J., & Leggett, T. (2016). Qualitative research: An introduction. *Radiologic Technology*, *88*(2), 225-231. Retrieved from http://www.asrt.org

Yazan, B. (2015). Three approaches to case study methods in education: Yin, Merriam, and Stake. *The Qualitative Report*, *20*(2), 134-152. Retrieved from http://nsuworks.nova.edu/tqr/

Ye, J., Zhang, B., Ruilin, L., Feng, C., & Tang, C. (2019). Program state sensitive parallel fuzzing for real world software. *IEEE Access*, *7*, 42557-42564. doi:10.1109/ACCESS.2019.2905744

Yi, J., Tan, S. H., Mechataev, S., Bohme, M., & Roychoudhury, A. (2018). A correlation study between automated program repair and test suite metrics. *In Proceedings of the 40th International Conference on Software Engineering ACM, 2*4. doi:10.1145/3180155.3182517

Yin, R. K. (2018). *Case study research and applications: Design and methods* (6th ed). Thousand Oaks, CA: Sage.

Yip, C., Han, N. L. R., & Sng, B. L. (2016). Legal and ethical issues in research. *Indian Journal of Anaesthesia*, *60*(9), 684-688. Retrieved from Walden University Databases

Yip, P. M., Venner, A. A., Shea, J., Fuezery, A., Huang, Y., Massicotte, L., … & Shaw, J. L. V. (2018). Point of care testing: A position statement from the Canadian society of clinical chemists. *Clinical Biochemistry*, *53,* 156-159. doi:10.1016/j.clinbiochem.2018.01.015

Yoon, K., Dols, R., Hulscher, L., & Newberry, T. (2016). An exploratory study of library website accessibility for virtually impaired users. *Library & Information Science Research*, *38*(3), 250-258. doi:10.1016/j.lisr.2016.08.006

Yu, L., Alegroth, E., Chatzipetrou, P., & Gorschek, T. (2020). Utilising CI environment for efficient and effective testing NFRs. *Information and Software Technology*, *117*, 1-19. doi:10.1016/j.infsof.2019.106199

Zachariah, B. (2015). Optimal stopping time in software testing based on failure size approach. *Annals of Operations Research*, *235,* 771-784. doi:10.1007/s10479-015-1959-5

Zalewski, J., & Gonzalez, F. (2017). Evolution in the education of software engineers: Online course on cyberphysical systems with remote access to robotic devices. *International Journal of Online Engineering*, *13*(8), 133-146. doi:10.3991/ijoe.v13i08.7377

Zein, S., Salleh, N., & Grundy, J. (2016). A systematic mapping study of mobile application testing technique. *Journal of Systems and Software*, *117*, 334-356. doi:10.1016/j.jss.2016.03.065

Zhang, X., Woud, M., Velten, J., Margraf, J., & Kuchinke, L. (2017). Survey method matters: Online/offline questionnaires and face-to-face or telephone interviews differ. *Computers In Human Behavior, 71*, 172-180. doi:10.1016/j.chb.2017.02.006

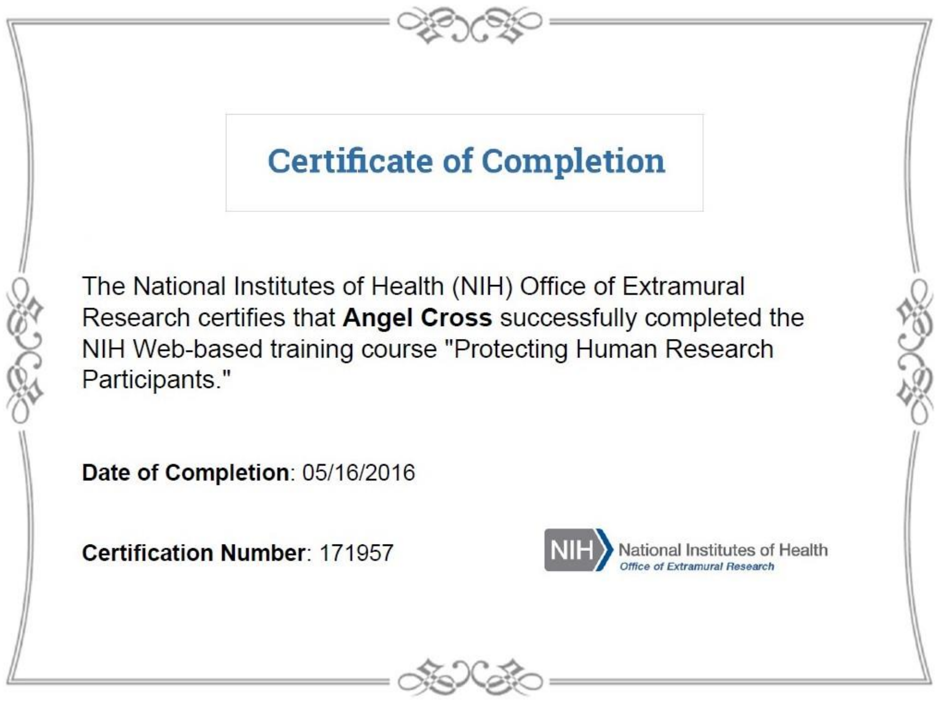
Zhao, M., & Chen, S. (2018). The effects of structured physical activity program on social interaction and communication for children with autism. *BioMed Research International*, *2018*, 1-13. doi:10.1155/2018/1825046

Zhi, J., Garousi, Y., Sun, B., Garousi, G., Shahnewaz, S., & Ruhe, G. (2015). Cost, benefits and quality of software development and documentation: A systematic mapping. *The Journal of Systems and Software*, *99*, 175-198. doi:10.1016/j.jss.2014.09.042

Zhou, Z. Q., Sinaga, A., Susilo, W., Zhao, L., & Cai, K. (2018). A cost-effective software testing strategy employing online feedback information. *Information Sciences*, *422*, 318-335. doi:10.1016/j.ins.2017.08.088

Zhu, M., & Pham, H. (2018). A software reliability model incorporating martingale process with gamma-distributed environmental factors. *Annals of Operations Research*, 1-22. doi:10.1007/s10479-018-2951-7

Appendix A: Human Subject Research Certificate of Completion



Certificate of Completion

The National Institutes of Health (NIH) Office of Extramural Research certifies that **Angel Cross** successfully completed the NIH Web-based training course "Protecting Human Research Participants."

Date of Completion: 05/16/2016

Certification Number: 171957

NIH National Institutes of Health
Office of Extramural Research

Appendix B: Interview Protocol

Interview Title: Exploring testing strategies to ensure the reliability of software applications in the government contracting industry

Participant ID: ______________ Date: _____________

Interview Mode: Phone or Skype Telephone: _____________ Starting Time: _____

A. I will introduce myself to the participant and thank him or her for their voluntary participation

B. I will verify receipt of the consent form and answer any questions or concerns the participant may have

C. I will remind the participant that the interview will be recorded, and the interview will remain confidential

D. I will begin recording announcing the participant’s anonymous code along with the date and time of the interview

E. I will start the interview with the first background question and continue through the process until the last question has been asked

1. Can you tell me about yourself and your current role?
2. How many years of experience do you have as a software developer?
3. How long have you been performing software testing tasks?
4. What type of project(s) are you currently working on?

F. I will start the interview with the first interview question and continue through the process until the last question has been asked

1. What is the primary software development methodology you are using?

2. How is software testing organized in your organization?
3. What testing strategies have you used to ensure the reliability of software applications?
4. How do you assess the effectiveness of the testing strategies used to ensure the reliability of software applications?
5. How satisfied are you with the development and testing environments that you have?
6. What challenge(s) have you faced where you find yourself in a disagreement over a software defect?
7. How has these challenges impacted your testing of software applications?
8. What testing strategies do you find the most efficient in detecting software defects?
9. How much time is allocated for testing software applications in your organization?
10. What additional information would you like to share about software testing that would ensure the reliability of software applications?

G. End the interview questions and ask if there is any additional information that they would like to add that might be applicable and that we did not discuss?

H. Thank the participant for participating in the study. Confirm that the participant has contact information for any follow-up questions or concerns.

Ending Time: ____________

Appendix C: Background/Interview Questions

Background Interview Questions

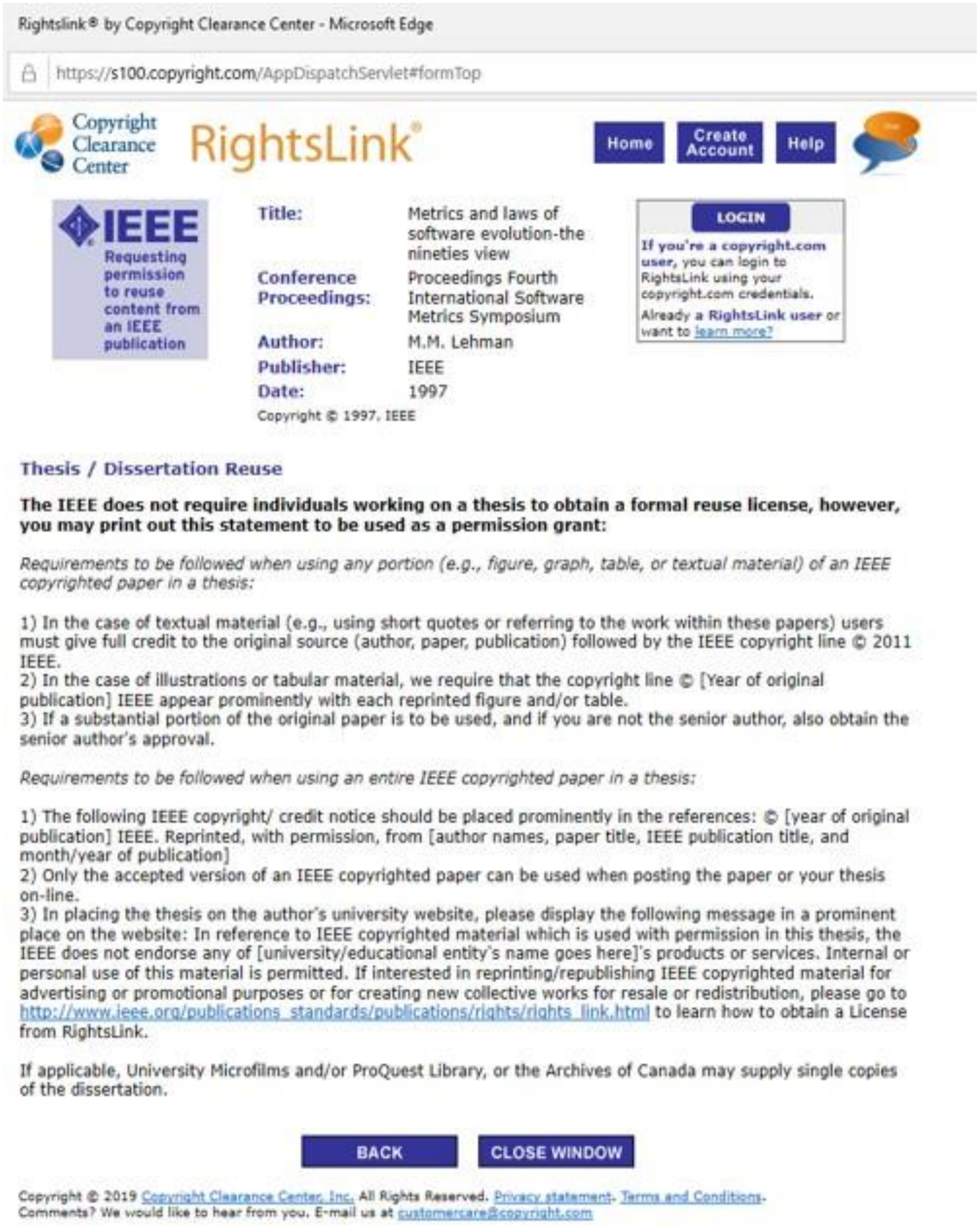
1. Can you tell me about yourself and your current role?
2. How many years of experience do you have as a software developer?
3. How long have you been performing software testing tasks?
4. What type of project(s) are you currently working on?

Interview Questions

1. What is the primary software development methodology you are using?
2. How is software testing organized in your organization?
3. What testing strategies have you used to ensure the reliability of software applications?
4. How do you assess the effectiveness of the testing strategies used to ensure the reliability of software applications?
5. How satisfied are you with the development and testing environments that you have?
6. What challenges have you faced where you find yourself in a disagreement over a software defect?
7. How has these challenges impacted your testing of software applications?
8. What testing strategies do you find the most effective in detecting software defects?
9. How much time is allocated for testing software applications in your organization?

10. What additional information would you like to share about testing strategies that would ensure the reliability of software applications?

Appendix D: Permission to Use Figures #1



Rightslink® by Copyright Clearance Center - Microsoft Edge

https://s100.copyright.com/AppDispatchServlet#formTop

Copyright Clearance Center
RightsLink®
Home
Create Account
Help
IEEE
Requesting permission to reuse content from an IEEE publication
LOGIN

Title:	Metrics and laws of software evolution-the nineties view
Conference Proceedings:	Proceedings Fourth International Software Metrics Symposium
Author:	M.M. Lehman
Publisher:	IEEE
Date:	1997

Copyright © 1997, IEEE

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.
Already a **RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

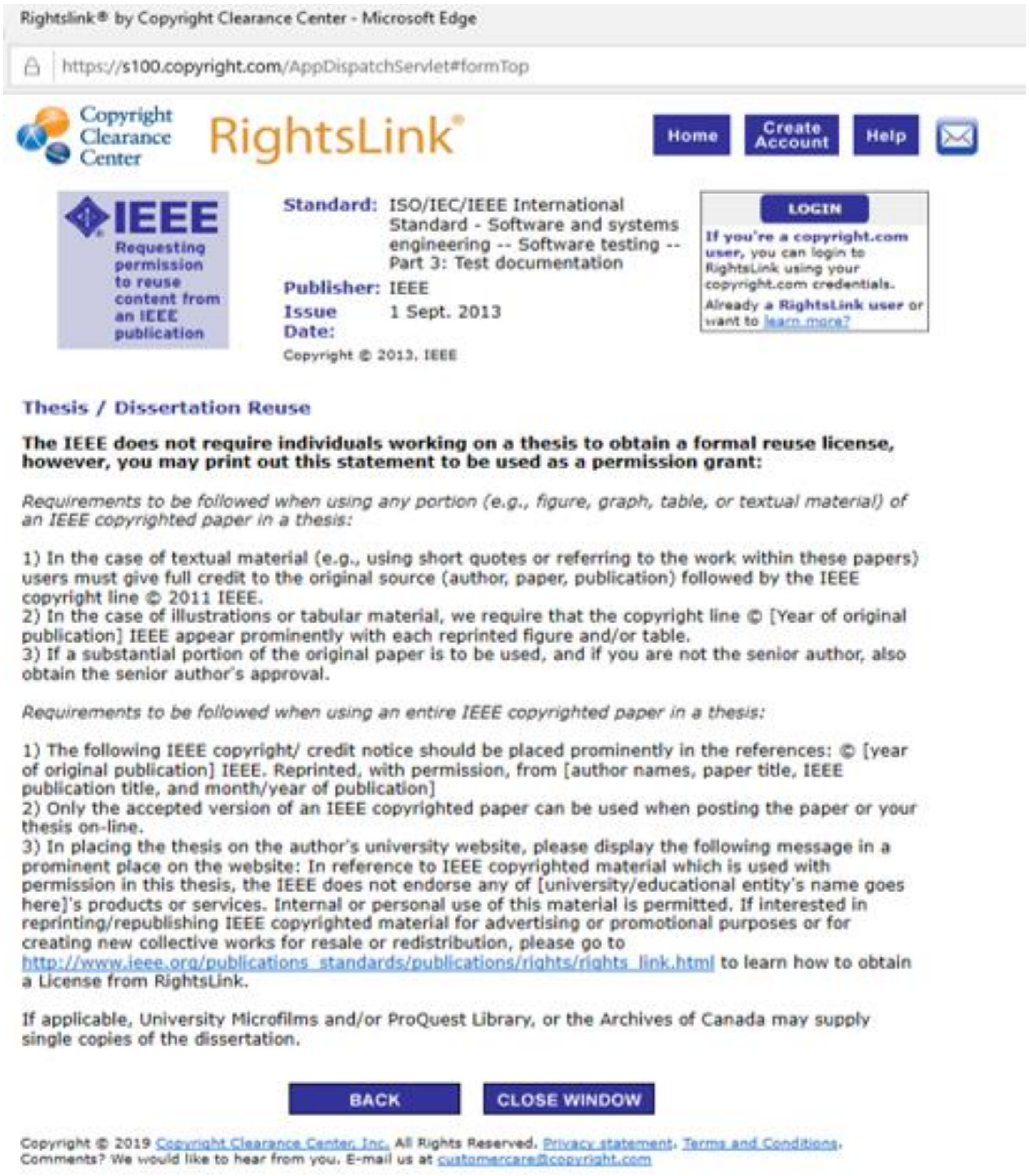
Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Copyright © 2019 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement. Terms and Conditions.
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Appendix E: Permission to Use Figures #2


Rightslink® by Copyright Clearance Center - Microsoft Edge
https://s100.copyright.com/AppDispatchServlet#formTop
Copyright Clearance Center
RightsLink®
Home
Create Account
Help
IEEE
LOGIN
BACK
CLOSE WINDOW

Requesting permission to reuse content from an IEEE publication

Standard: ISO/IEC/IEEE International Standard - Software and systems engineering -- Software testing -- Part 3: Test documentation
Publisher: IEEE
Issue Date: 1 Sept. 2013
Copyright © 2013, IEEE

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.
Already **a RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

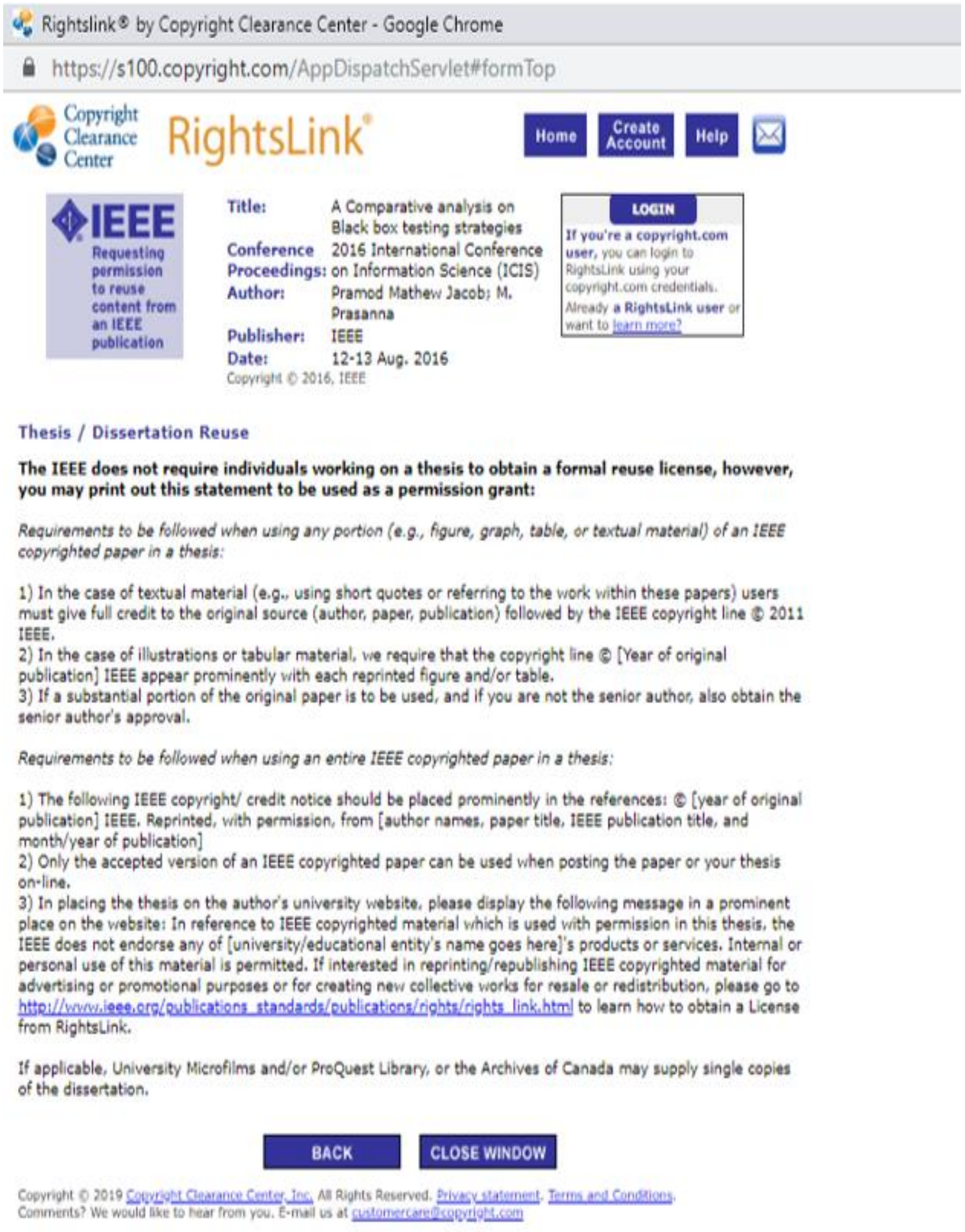
Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Copyright © 2019 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement. Terms and Conditions.
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Appendix F: Permission to Use Figures #3


Rightslink® by Copyright Clearance Center - Google Chrome
https://s100.copyright.com/AppDispatchServlet#formTop
Copyright Clearance Center
RightsLink
Home
Create Account
Help
IEEE
Requesting permission to reuse content from an IEEE publication
LOGIN
BACK
CLOSE WINDOW

Title: A Comparative analysis on Black box testing strategies
Conference Proceedings: 2016 International Conference on Information Science (ICIS)
Author: Pramod Mathew Jacob; M. Prasanna
Publisher: IEEE
Date: 12-13 Aug. 2016
Copyright © 2016, IEEE

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.
Already **a RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

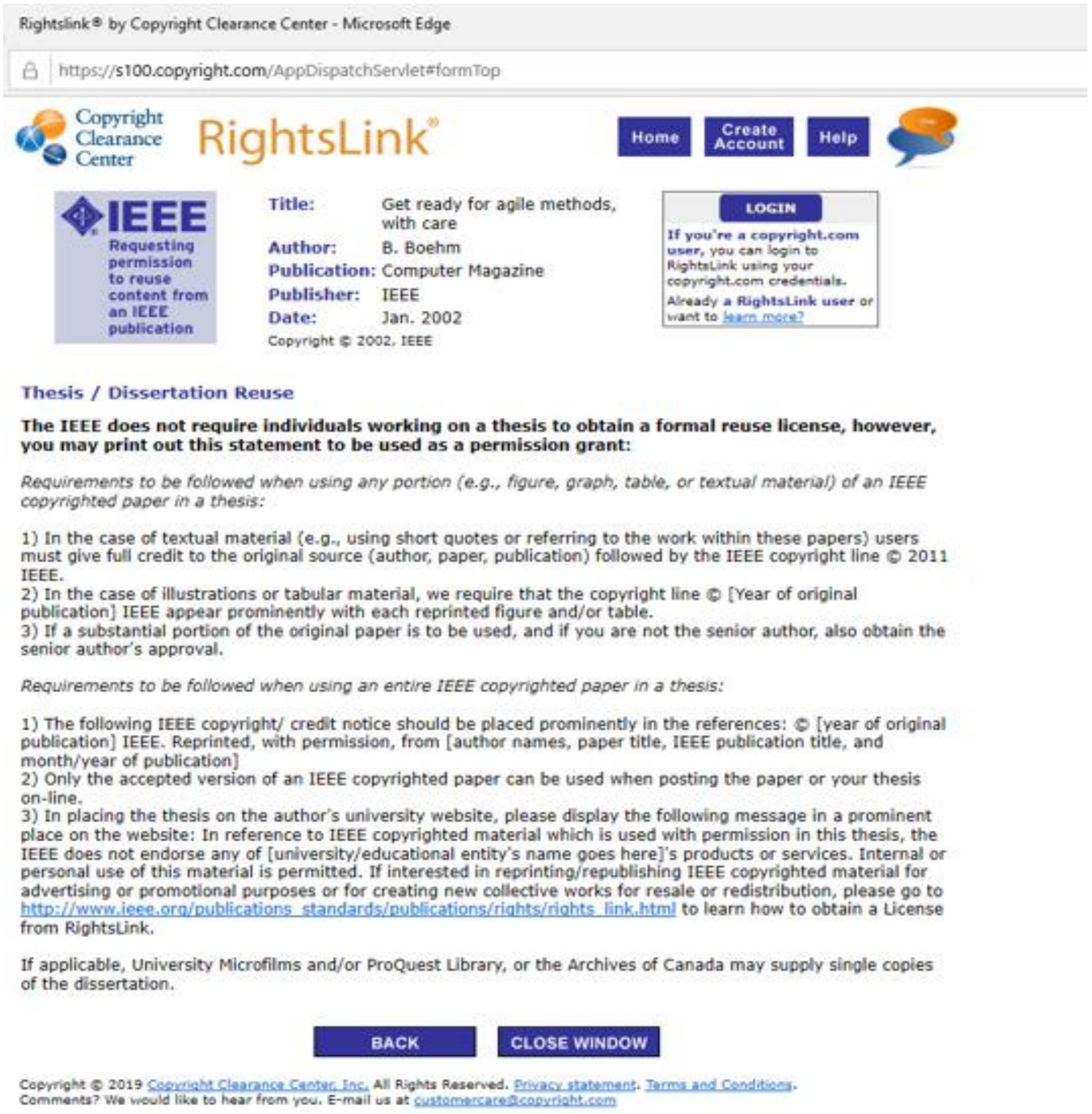
Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Copyright © 2019 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement. Terms and Conditions.
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Appendix G: Permission to Use Figures #4


Rightslink® by Copyright Clearance Center - Microsoft Edge
https://s100.copyright.com/AppDispatchServlet#formTop
Copyright Clearance Center
RightsLink®
Home
Create Account
Help
IEEE
Requesting permission to reuse content from an IEEE publication
LOGIN

Title: Get ready for agile methods, with care
Author: B. Boehm
Publication: Computer Magazine
Publisher: IEEE
Date: Jan. 2002
Copyright © 2002, IEEE

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.
Already a RightsLink user or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

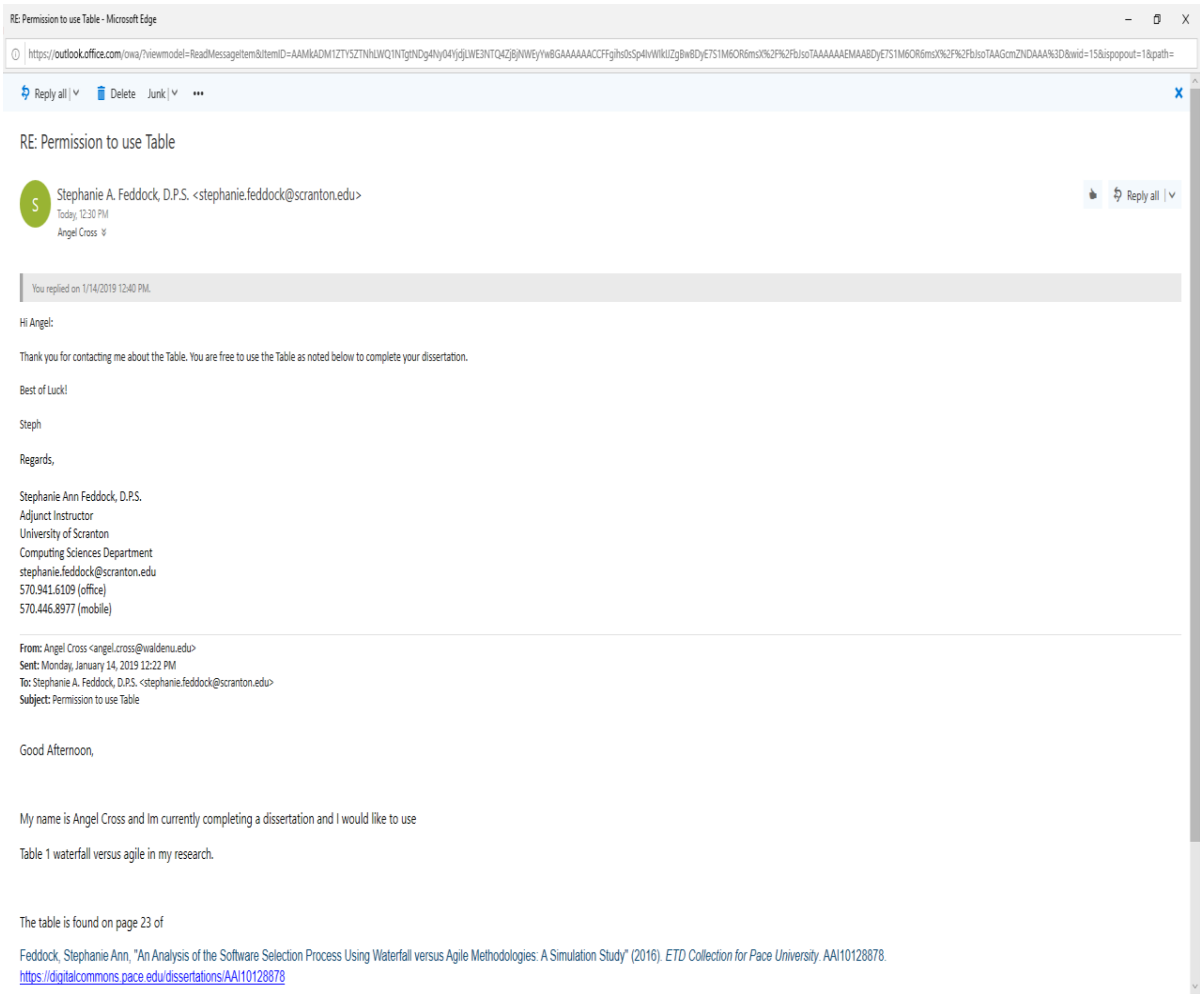
Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Copyright © 2019 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement. Terms and Conditions.
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Appendix H: Permission to Use Figures #5



RE: Permission to use Table - Microsoft Edge

https://outlook.office.com/owa/?viewmodel=ReadMessageItem&ItemID=AAMkADM1ZTY5ZTNhLWQ1NTgtNDg4Ny04YjdjLWE3NTQ4ZjBjNWEyYwBGAAAAAACCFFgihs0sSp4lvWIkIJZgBwBDyE7S1M6OR6msX%2F%2FbJsoTAAAAAAEMAABDyE7S1M6OR6msX%2F%2FbJsoTAAGcmZNDAAA%3D&wid=15&ispopout=1&path=

RE: Permission to use Table

Stephanie A. Feddock, D.P.S. <stephanie.feddock@scranton.edu>
Today, 12:30 PM
Angel Cross

You replied on 1/14/2019 12:40 PM.

Hi Angel:

Thank you for contacting me about the Table. You are free to use the Table as noted below to complete your dissertation.

Best of Luck!

Steph

Regards,

Stephanie Ann Feddock, D.P.S.
Adjunct Instructor
University of Scranton
Computing Sciences Department
stephanie.feddock@scranton.edu
570.941.6109 (office)
570.446.8977 (mobile)

From: Angel Cross <angel.cross@waldenu.edu>
Sent: Monday, January 14, 2019 12:22 PM
To: Stephanie A. Feddock, D.P.S. <stephanie.feddock@scranton.edu>
Subject: Permission to use Table

Good Afternoon,

My name is Angel Cross and Im currently completing a dissertation and I would like to use

Table 1 waterfall versus agile in my research.

The table is found on page 23 of

Feddock, Stephanie Ann, "An Analysis of the Software Selection Process Using Waterfall versus Agile Methodologies: A Simulation Study" (2016). *ETD Collection for Pace University*. AAI10128878.
https://digitalcommons.pace.edu/dissertations/AAI10128878