A FRAMEWORK FOR IMPROVING PERFORMANCE TESTING IN AGILE SOFTWARE DEVELOPMENT

by

Erik Whiting, BS

THESIS

Presented to the Faculty of

The University of Houston-Clear Lake

In Partial Fulfillment

Of the Requirements

For the Degree

MASTER OF SCIENCE

in Software Engineering

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

DECEMBER, 2020

A FRAMEWORK FOR IMPROVING PERFORMANCE TESTING IN AGILE SOFTWARE DEVELOPMENT

by

Erik Whiting

APPROVED BY

____
Soma Datta, PhD, Chair

____
James Carlton Helm, PhD, Committee Member

____
James B. Dabney, PhD, Committee Member

RECEIVED/APPROVED BY THE COLLEGE OF SCIENCE AND ENGINEERING:

____
David Garrison, PhD, Interim Associate Dean

____
Miguel A. Gonzalez, PhD, Dean

Dedication

For every stakeholder in software quality.

Acknowledgements

I want to thank all the wonderfully supportive professors in the College of Science and Engineering who have helped me in the last two years. I also want to thank my wife who has been endlessly and inexplicably patient with me throughout my academic career.

ABSTRACT

A FRAMEWORK FOR IMPROVING PERFORMANCE TESTING IN AGILE SOFTWARE DEVELOPMENT

Erik Whiting
University of Houston-Clear Lake, 2020

Thesis Chair: Soma Datta, PhD

The subdiscipline of software quality assurance concerned with non-functional requirements (NFRs) and hardware metrics is known as performance testing. Conducting effective performance testing is complicated, time consuming, and expensive. These attributes put performance testing at odds with agile software development methodologies, which incrementally build software systems in quick cycles while being supported by exhaustive unit and integration test coverage. Due to a variety of challenges, performance testing often cannot keep up with an agile release cadence, and there is a growing body of research that catalogues and describes these challenges and proposes solutions to some of them. This study presents a software testing framework which implements several of the proposed solutions. The framework, called Lulu Performance Test (LPT), aims to confront many of the challenges noted in recent research, with the goal of making effective performance testing more palatable to agile software development methodologies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I:
AGILE SOFTWARE DEVELOPMENT AND PERFORMANCE TESTING

Agile and Performance Testing Defined

Agile software development (ASD) [1], when compared to traditional development methodologies, is relatively new to software engineering. ASD is actually a collection of methodologies which share twelve principles [2] [3] [4]. Methodologies considered ASD share the themes of frequent delivery of working software, constant technical excellence, consistent velocity, and good communication. The actualization of these principles is achieved through shared practices common to the different methodologies, most relevant to this paper of which include dedication to effective software testing.

Testing is a broad discipline in software engineering [5], and its specific implementation depends upon the software development methodology being used [6] [7]. There are two broad types of testing: black box, where testers have no insight into the inner workings of a system such as source code, database schemas, and other design documents; and white box, where testers do have access to the formal specifications and implementations of the system. As an example, unit testing—tests that check atomic units of functionality such as individual functions and classes—are written in source code, typically the same kind of source code the system is built in, are considered white box tests since the source code itself is being tested. In contrast, usability testing—activities taken to evaluate how easily a system is used by its intended audience—does not require access to source code and is thus considered black box testing.

There are many kinds of testing in addition to those mentioned above. Integration testing involves activities to check that disparate parts of a system work together as expected and, depending on how it is conducted, could be accomplished as either a black box or white box technique. Exploratory testing involves probing a system from several angles, creatively, and without a clear metric to be evaluated. This kind of testing is strictly a black box testing activity. Smoke testing or sanity checking involves running a small number of tests against a system to ensure the most basic pieces of functionality are operating as expected and combines both white box and black box elements. System testing involves evaluating an entire system's adherence to its specifications and is a black box testing activity.

Performance testing is another kind of testing activity that can be conducted as both white box and black box methods, and is a blanket term for many different kinds of testing that measure a software system's hardware metrics under different conditions [8]. Test activities that are considered performance testing include load, stress, spike, endurance, volume, and scalability testing [9] [10] [11] [12] [13] [14]. Performance testing asserts traits of a system such as reliability, scalability, elasticity, security, and more. These traits are often colloquially called "the -ilities" and thus performance testing can be called "-ility testing." Other names for performance testing include non-functional requirements (NFR) testing, as well as quadrant 4 (Q4) testing, in reference to the Agile testing quadrants [15]. Like other kinds of testing, the specific ways in which performance testing is conducted vary depending on the development methodology being used to build the system in question.

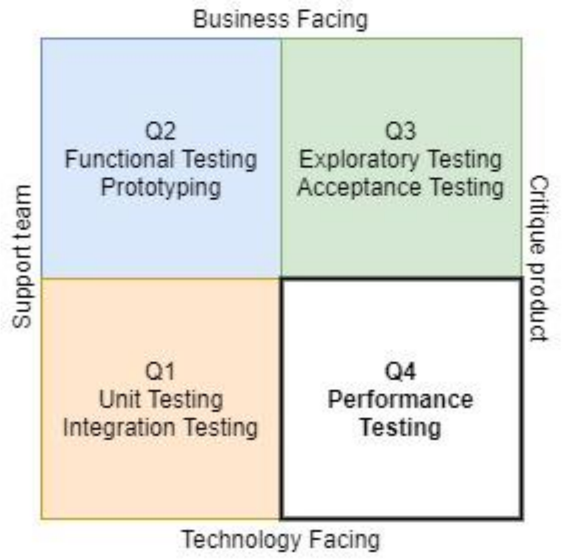

Figure 1.1
Agile Testing Quadrants by Brian Marick.

Performance Testing Challenges in Agile Software Development

Performance testing is a complicated, expensive, and time-consuming endeavor [16]. Its complexity comes from the fact that many of the metrics to be gathered for effective performance testing and meaningful analysis sit at the union of the software/hardware relationship. Its expense is due to several factors, including the high expense of experienced performance engineers, the hardware or cloud resources required to conduct performance testing effectively, and the expense of existing tools [17]. Finally, performance testing is time consuming, because it involves simulating users over an extended period of time and gathering relevant hardware metrics and then conducting complex analysis on those metrics. The time consuming and complex nature of performance testing also add to its cost [18].

Due to the challenges listed above, performance testing is not easily implemented in an ASD environment. The metrics gathered from performance testing are most meaningful when the environment in which they are gathered is most like the production environment intended for the system under test (SUT) [19]. In other words, performance testing requires a completely integrated and nearly finished system in order to provide accurate and meaningful insights. In contrast, ASD methodologies favor the delivery of working software in several successive and short increments. This means that many times, a system delivered by an agile team is never truly a completed system. Therefore, the metrics gathered on an SUT developed by an agile team at least have the potential to be misleading, if not completely false. Additionally, the release cadence of a system built in an ASD environment requires quick feedback from tests. As mentioned before, performance testing is a time consuming endeavor, which makes it a hindrance to Agile methodologies [16] [20].

Purpose of This Study

Nearly all industries which rely on application development can benefit from adopting Agile software development (ASD) methodologies. Paramount to a system's success in any development methodology is the quality of the system's tests, and the feedback they provide [20]. Because of the challenges around performance testing in ASD, improvements to this practice will improve agile software development. The immediate purpose of this study is to identify promising and complementary techniques regarding performance testing in the recent literature and consolidate them into a single testing framework for both professional and academic software engineers. More broadly, the long-term goal of this study is to improve upon the quality of software developed by software engineers everywhere. Software pervades the daily lives of nearly all people around the world, improvements to its quality hopefully lead to improvements to the overall quality of life.

The framework developed will comprise of techniques from the recent literature and integrate easily into development processes to provide the most benefit to intended users. The goals of this framework are to provide an easy to use, open source, and technology-agnostic tool to facilitate performance testing in ASD. The design and development of this framework--named Lulu Performance Test, or just LPT--will therefore make trade-offs which favor these traits. The intended audience of the framework is software developers and quality assurance personnel working in an ASD environment who prefer tools that are both ready to use and easy to augment. Given this scope, the system will be open source and free to use.

CHAPTER II:
LITERATURE REVIEW

The purpose of this thesis and the software artifact developed alongside it is to improve the current state of software performance testing in agile software development (ASD). Given the scope detailed in the previous chapter, the goals of this project will best be achieved by focusing on a few specific challenges and building upon concepts already presented in the research instead of developing solutions from scratch. To guide this research and the development of the software artifact, a systematic literature review was conducted by the author and his thesis advisor.

Systematic Literature Review Research Questions

To ensure the literature review aided the goals of this research, the following systematic literature review research questions were developed:

- **SLR-RQ1.** What are the current ideas, challenges, and solutions presented in the peer-reviewed literature regarding performance testing and Agile software development?
- **SLR-RQ2.** What trends are emerging in the current peer-reviewed literature regarding performance testing and Agile software development?

The rest of this chapter will detail the methods used to answer these questions and the results found, part of Chapter IV will discuss how the recent literature informed the design of the Lulu Performance Test (LPT) framework.

The goal in answering the first research question is to provide context for readers and future researchers. Sharing the common ideas existing in the research will provide a common understanding and vocabulary when discussing the findings of this and future research. Additionally, understanding the common challenges help consolidate research efforts by promoting a shared understanding among those working in this area. Finally, a shared understanding of the current proposed solutions promotes the minimization of duplicated efforts, allowing researchers to know what has been attempted, what has worked, and what ideas need more development.

In answering the second research question, researchers may be able to identify trends that will lead to favorable results. By showing the current trends in the literature related to the common challenges in performance testing in Agile software development, ideas worth developing further are easier to spot, and new progress can be made. Furthermore, in highlighting current trends, it will also be easier to identify solutions that have not yet been attempted in the existing research, fostering innovation among solutions to the common problems.

To ensure the systematic literature review was reproducible by future researchers or other interested parties, the steps and analysis will be presented in this section.

Literature Review Steps

The overall structure and method of this study is influenced by Vallon et. al [21]. As such, the research steps followed in this study are very similar. They are as follows:

1. *Planning the review:*
   a. Identifying areas of interest to review
      i. Selecting this area (performance testing and Agile)
   b. Developing research questions
   c. Developing review protocol
2. *Conducting the review:*
   a. Locating recent literature
   b. Filtering of literature
   c. Data extraction and analysis
3. *Report findings*

Literature Search

In order to locate the appropriate resources for the subject at hand, the search terms are designed to find all relevant and recent literature. Because many software development methodologies fall under the banner term “Agile,” several search terms had to be combined to encompass Agile development, and this report uses the same terms for "Agile" as used in the report by Vallon et. al [21]. Additionally, "performance" testing can go by many names and the authors made the best attempt at finding a list of terms that would exhaustively cover all of performance testing subjects. The query used to yield the papers for this study is generalized as follows:

```
(agile OR scrum OR "extreme programming" OR "pair programming" OR "lean
software development" OR "iterative development")
AND
("performance test" OR "performance engineering" OR "non-functional
test" OR "NFR test")
```

Figure 2.1
Search Terms

The search terms are applied to both the full text and metadata. This study applies these search terms to the ACM Digital Library, IEEE Xplore, and Web of Science databases. Search queries are configured slightly differently for each database and the following sections document the exact queries used in each database.

Deciding Which Studies are Relevant

After querying a database with the above search terms, the results were filtered twice. First, the results are filtered to include only the literature published between the beginning of 2015 and the end of 2019, effectively gathering research published in the five years prior to 2020. This search was conducted in June of 2020; however, this is not relevant due to the fact the December 2019 publications are the most recent items gathered. This is to ensure that only the most recent and relevant literature is analyzed for this study. Secondly, to ensure that the most authoritative sources are evaluated, the results are filtered once again to include only journal articles and conference proceedings. The next section explains how research items are further analyzed for fitness in this work.

Search Terms by Database

The search terms mentioned earlier were applied to three different databases: IEEE Xplore, ACM Digital Library, and Web of Science. After applying the filters described in previous section, there are 117 search results altogether. The next three sub-sections describe the steps taken on each individual search database, so the results can be duplicated.

IEEE Explore Database Search

Once on the IEEE Xplore homepage, click "Command Search" and fill in the subsequent box with this specific query:

```
(
"Full Text Only": "agile"
OR "Full Text Only": "scrum"
OR "Full Text Only": "extreme programming"
OR "Full Text Only": "pair programming"
OR "Full Text Only": "lean software development"
OR "Full Text Only": "iterative development"
) AND (
"Full Text Only": "performance test"
OR "Full Text Only": "non-functional test"
OR "Full Text Only": "NFR test"

)
```

Figure 2.2
IEEE Search Terms

And click "search." On the next page, find the "Year" range selector and set the years to be from 2015 to 2019 and click "apply." Finally, check the "conferences" and "journals" check box. Altogether, there should be 80 results. These are the papers that will be further analyzed in "Title Review" and "Abstract Review" detailed in later sections.

ACM Digital Library Database Search

Once on the ACM Digital Library homepage, click “Advanced Search.” On the next page, select “Full Text” from the “Search Within” drop down. In the search terms box, enter the following query:

```
("agile" OR "scrum" OR "extreme programming" OR "pair programming" OR
"lean software development" OR "iterative development")
AND
("performance test" OR "non-functional test" OR "NFR test")
```

Figure 2.3
ACM Search Terms

Scroll down to the “Publication Date” section and select “Custom Range” and set the range to be from January 2015 to December 2019. Click “search.” On the next page, in the “Publications” section, click the “All Publications” drop down and select “Proceedings” (there should be 35). After this, go back, and select “Journals” from the “Publications” sub-section “All Publications” (there should be 2). These 37 papers will be further filtered in the Title and Abstract Reviews described in a later section.

Web of Science Database Search

Once on the Web of Science homepage, click “Advanced Search” and put the following query into the search box:

```
AB=("agile" OR "scrum" OR "extreme programming" OR "pair programming"
OR "lean software development" OR "iterative development")
AND
AB=("performance test" OR "non-functional test" OR "NFR test")
```

Figure 2.4
Web of Science Search Terms

Note that Web of Science does not have the option to search by full text, so we are searching the abstracts.

Under “Timespan” select “Custom year range” and set the range to 2015 to 2019. Click “search.” There will be zero results from this search.

Research Item Quality Review

After exporting the bibliographies and downloading the search results, the research items were put through a quality review to ensure they were relevant to this research. The quality review involved a title review, abstract review, and article review, which will be expounded upon in further detail in the following subsections. If at any point of the quality review a research item was deemed as irrelevant to this study, it was removed from the list of results and was not part of the final analysis.

Title Review

After the search results were filtered, their bibliographic information was exported and consolidated into an Excel spreadsheet. On this spreadsheet, each author of this study evaluated the title of every search result and annotated whether they thought it is relevant to this study or not. When the authors disagreed on the potential relevance, the authors discussed their disagreement until a consensus was reached. If a search result was decided to be irrelevant based on its title, the authors removed it from the list of relevant studies. Appendix C lists the studies deemed irrelevant to this report.

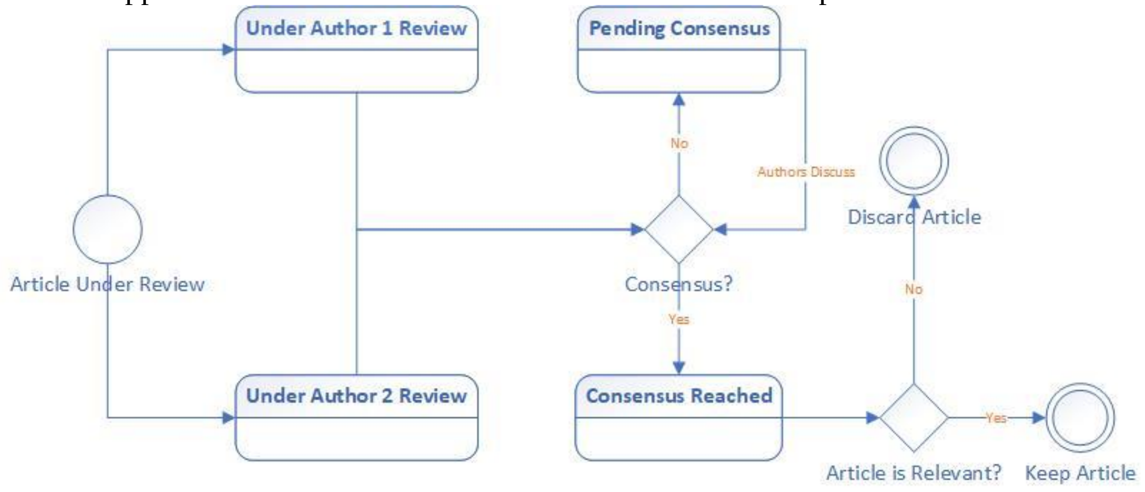

Figure 2.5
Article Review Process Workflow

Altogether, 42 papers were removed during title review: 28 from the IEEE Xplore result set, and 14 from the ACM Digital Library result set. After title review, the remaining 75 results were evaluated for relevance via abstract review.

Abstract Review

After title review, the remaining results were evaluated based on their abstracts. Papers were considered irrelevant to this study if the abstracts conveyed that the papers would not talk about either software performance testing or testing in Agile software development. Like in title review, the authors evaluated and discussed their opinions on the potential relevance of the search results until they came to a consensus. Appendix C lists the studies the authors deemed irrelevant to this report.

Altogether, 21 papers were removed: 19 from the IEEE Xplore result set, and 2 from the ACM Digital Library result set. The remaining 54 papers were then checked for duplicates, of which, 2 were found. The studies marked as duplicate were removed from the ACM collection of papers and ultimately, 52 papers are selected for article review.

Article Review

After abstract review, the remaining 52 articles were read in their entirety to be evaluated for relevance to this study. Like the title review and abstract review phases, the authors of this report evaluated and discussed whether each article was relevant to this study. Once a consensus was reached, the articles were deemed relevant and passed the final phase of the review protocol.

Altogether 13 papers were removed: 9 from the IEEE Xplore result set, and 4 from the ACM Digital Library result set. Finally, there are 39 studies analyzed in this report.

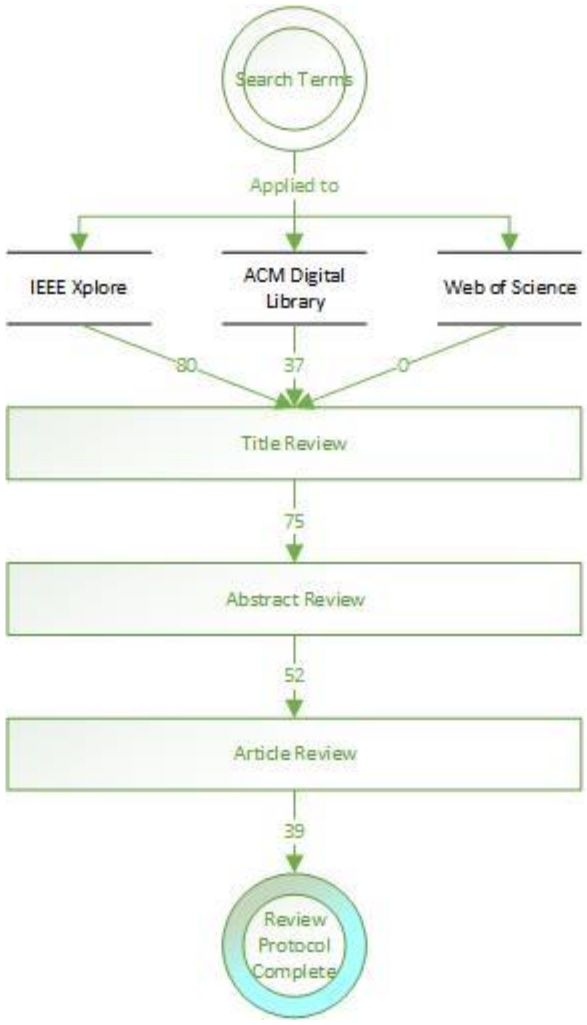

Figure 2.6
Article Review Process

After performing the quality assessment on the literature as described above, some patterns and categories began to emerge. The rest of this chapter will annotate and expound on those patterns. First, the various research items can be categorized by their content in one of three types. Secondly, papers that propose solutions can be categorized by broad solution types. Finally, many papers cite different challenges and obstacles related to either performance testing or Agile software development (and sometimes both), which can also be broadly categorized. The next several subsections will detail the categorizations of the papers analyzed in this study.

Content Type Categories

After reviewing the literature related to the proposed research questions, it was found that research items broadly fell into one of three categories:

- **Solution Proposals:** Papers designated as solution proposals are those that present a new way of conducting performance testing or approaching testing in an Agile context in response to a specific challenge. Solutions proposed can include the introduction of a new framework or technique, or they can include a rethinking of existing tools and processes. Papers are considered solution proposals if they present something like a new tool, method, or algorithm in response to a stated challenge relevant to the research questions in this paper.
- **Reports:** Papers are considered reports when they document the experiences or findings related to performance testing. These papers do not necessarily present a solution to a found problem, they simply provide some useful knowledge to the research questions being asked in this study. An example of a research item designated as a “report” is a paper that gathered and analyzed data from a group of Agile developers related to their feelings towards performance testing.
- **Literature Reviews:** A third category which merits its own designation is that of literature reviews. Within the search results defined in sections 3.4 and 3.5, there were several studies such as this one. The literature reviews which passed the described quality assessments were deemed to contain valuable information related to this research even though none of them seek to answer the same exact questions.

The categories listed above may offer limited insight into the current state of performance testing and Agile software development, but they are categorized in this way to offer further analysis. For example, we will analyze solution papers for the types of solutions they offer.

Solution Type Categories

Solution proposals are further categorized into one of seven broad solution types. The following categories are necessarily broad to accommodate the creativity and ingenuity of proposed solutions in the literature while still allowing some kind of analysis to be conducted. As such, the authors urge readers to keep in mind that each solution proposal in the literature offers a unique or novel way to approach a problem despite its categorization here. The solution types for this study were captured as follows:

Model based solutions: Papers counted as model-based solutions are presenting solutions that implement model-based testing (MBT) [22] [23] to address the challenges of performance testing. MBT is a testing paradigm which focuses on a more abstract level of testing than traditional unit tests. These papers put forth augmentations to MBT processes that favor performance testing.

Test Automation: Some papers put forth solutions that generate code or automated tests. The code generated is usually a test script or load definition. To be considered a "test automation" solution, the paper had to propose a solution which automatically generated some artifact or artifacts related to performance testing.

Load manipulation: Some papers in the search results present solutions in which the workload of a performance test is generated. An example is a program dynamically calculating the optimum number of simulated users to run through test scripts. Another example is one paper which injected realistic "think times" into test script executions.

Domain specific languages (DSLs): Papers falling into this category either introduce a DSL or augment an existing one. In this research, DSLs are always proposed as supporting tools to a more comprehensive solution in the paper. In other words, DSLs are never presented as solutions themselves, but as a tool for driving the proposed solution.

Visualization based solutions: These solution types offer ways of presenting data gathered from test runs in some novel way. These solutions endeavor to give stakeholders a new or unique point of view to some aspect of performance engineering activities.

Continuous Integration/Continuous Delivery Extensions: Several papers introduce solutions that require a continuous integration or continuous delivery (CI/CD) platform to actualize. For example, monitoring solutions may require data gleaned from a CI/CD process to present findings, or a load manipulation tool may need to work in the context of a build server to effectively carry out its purpose.

Fault Injection Solutions: One paper did not neatly fall into any of the previous categories and thus merited its own solution type: fault injection. This has been included as a solution type not only because it is a unique kind of approach, but also because it seems to be a promising area for further research. As mentioned before, only one paper fell into this category.

Note that papers proposing a solution can be counted in more than one category. For example, a paper proposing a domain specific language for load manipulation is counted as both a DSL solution and load manipulation solution.

Challenge Type Categories

Because this report aims to identify the trends in performance testing solutions as well as the challenges present in performance testing in Agile software development, we also analyze the presence of these challenges in the literature. The papers analyzed for this study mention challenges in both the performance testing on its own and in Agile; as such, these challenges were analyzed separately.

Performance Testing Challenges

The literature presents several challenges specific to performance testing, all of which fall into one of nine categories. Like with solution types, papers can mention more than one type of challenge. The categories of performance testing challenges are as follows:

1. Accuracy – Related to how much test results reflect reality
2. Automation – These are problems around the limitations of test automation
3. Complexity – These are challenges related to the complexity of performance testing

4. Cost – Obstacles presented related to the expense of performance testing
5. Ignored – These challenges relate to software teams ignoring performance testing
6. Infrastructure – Challenges related to provisioning infrastructure and setting up environments
7. Load – These challenges relate to finding proper load configurations in performance tests
8. Resources – This category relates to the challenge of having proper tools
9. Time – This category is related to the time-based challenges of performance testing

Agile Testing Challenges

The literature also poses challenges that are specific to performance testing in Agile software development environments. These challenges are not mentioned as often as performance testing challenges but they were nonetheless categorized. Performance testing challenges specific to Agile development fall into one of the following categories:

1. Artifact Maintenance – Related to keeping test-related documentation up to date
2. Communication – These are challenges related to communication among teams and individuals
3. Collaboration – Related to the challenges of coordination performance testing tasks
4. Company Culture – Challenges around the testing culture of organizations
5. Lack of performance testing emphasis – Occurs when teams do not prioritize NFR testing
6. Lack of research – Challenges related to a lack of related literature
7. Technical Debt – Related to maintainability requirements that get put off for one reason or another
8. Time – Related to the time constraints present in Agile software development

Application Domain Categories

One final area of analysis is the application domain. Some research items from the literature search are specific to certain application domains. It is a small subset of papers that mention specific application domains, but these domains are recorded here anyway. The application domains mentioned in the literature are listed below.

- Big Data
- Blockchain
- Cloud
- Embedded systems

- IoT
- Microservices
- Software Product Lines (SPLs)
- Web applications
- Mobile applications

Analysis of Results

As mentioned in a previous section, analyzing the 39 papers revealed multiple patterns. These patterns are illustrated in the next subsections. For further information on how the selected papers were analyzed, refer to Appendix A for a list of the articles selected for analysis, or Appendix B for categorization information regarding the 39 papers.

Quantitative Analysis

Paper Type

The "Paper Type" refers to whether the paper presents a solution, report, or a literature review. Note that some papers can fall into multiple categories. Altogether, 22 papers present solutions, 14 present reports, and there are three literature reviews.

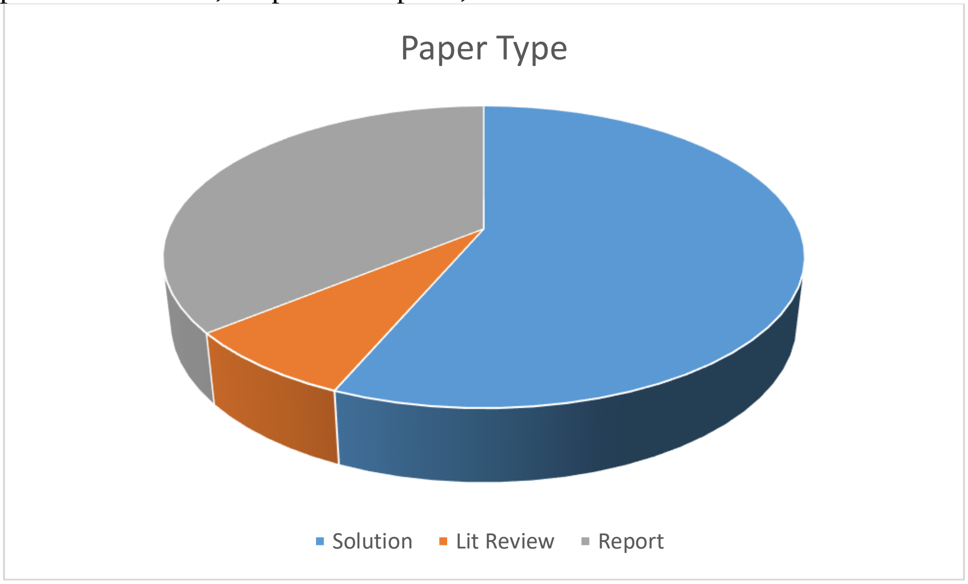

Figure 2.7
Paper Types

By year, there seven papers published in each year of 2015 through 2018, and 11 papers published in 2019. 2019 was the year in which both solution and literature reviews were most prevalent. 2016 saw the least amount of solution proposals, but the most reports.

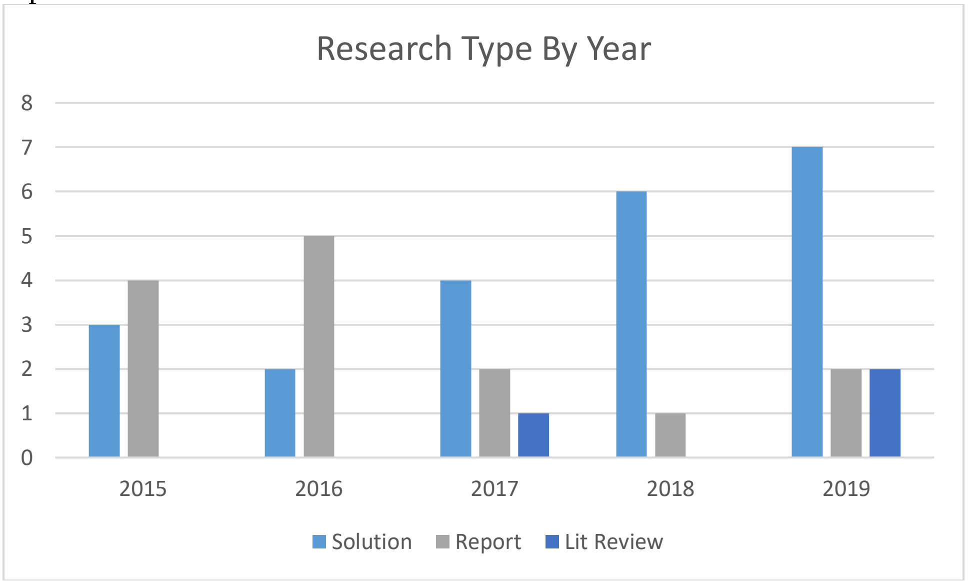

Figure 2.8
Paper Types by Year

Solution Types

When it comes to solution types, load manipulation was the most popular, followed by CI/CD based solutions, which was then followed by a three-way tie between DSL, visualization, and MBT based solutions.

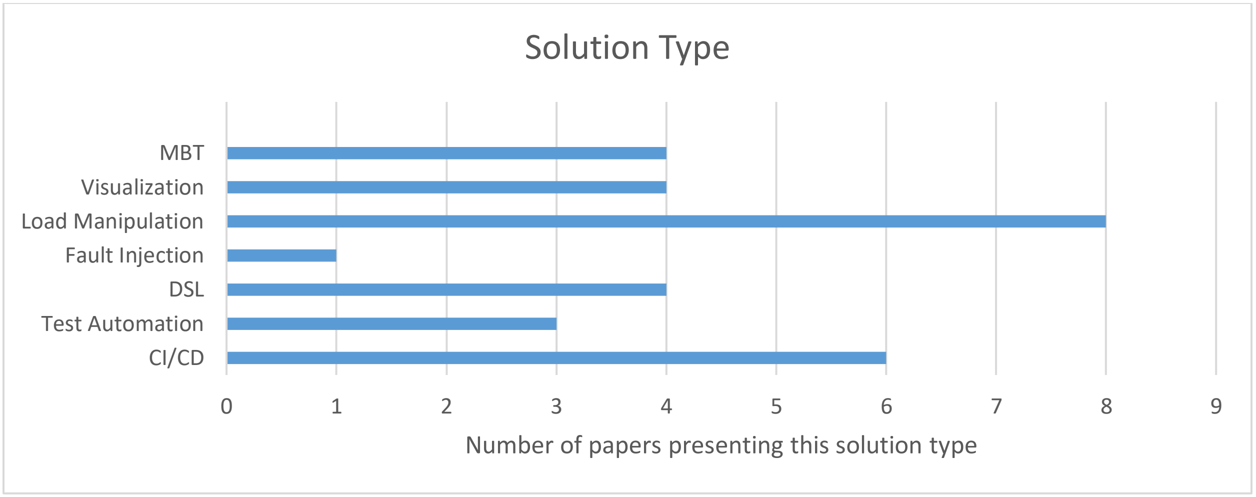

Figure 2.9
Solution Type Distribution

The next figure shows how solution types are grouped together. In other words, when papers present solutions that comprise of more than one category, the following figure is meant to show which types seem to complement each other best. Marginally, it appears load manipulation and DSL solutions go together most often in the research. Beyond this relationship though, there does not appear to be a statistically significant correlation between solution types. (Fault injection was neglected from the following table since it had no correlation)

	CI/CD	**DSL**	**Load Manipulation**	**Visualization**	**MBT**	**Test Automation**	**Log Evaluation**
CI/CD	x		1	1	1		
DSL		x	2		1	1	
Load Manipulation	1	2	x		1	1	1
Visualization	1			x			
MBT	1	1	1		x		
Test Automation		1	1			x	
Log Evaluation			1				x

Table 2.1
Solution Type Correlation

In addition to the types of solutions presented, we can also group solution types by years presented in order to identify trends in the research. Figure 8 visualizes this data. Note, in the visualization, we removed the "Fault Injection" solution type since it was only mentioned in one article and thus could not possibly show a trend. Looking at the visualization, it appears that load manipulation and CI/CD extensions are consistently popular solutions to performance testing problems. The visualization also seems to imply that the other categories of solution proposals are sporadically attempted as tools to solve performance testing problems.

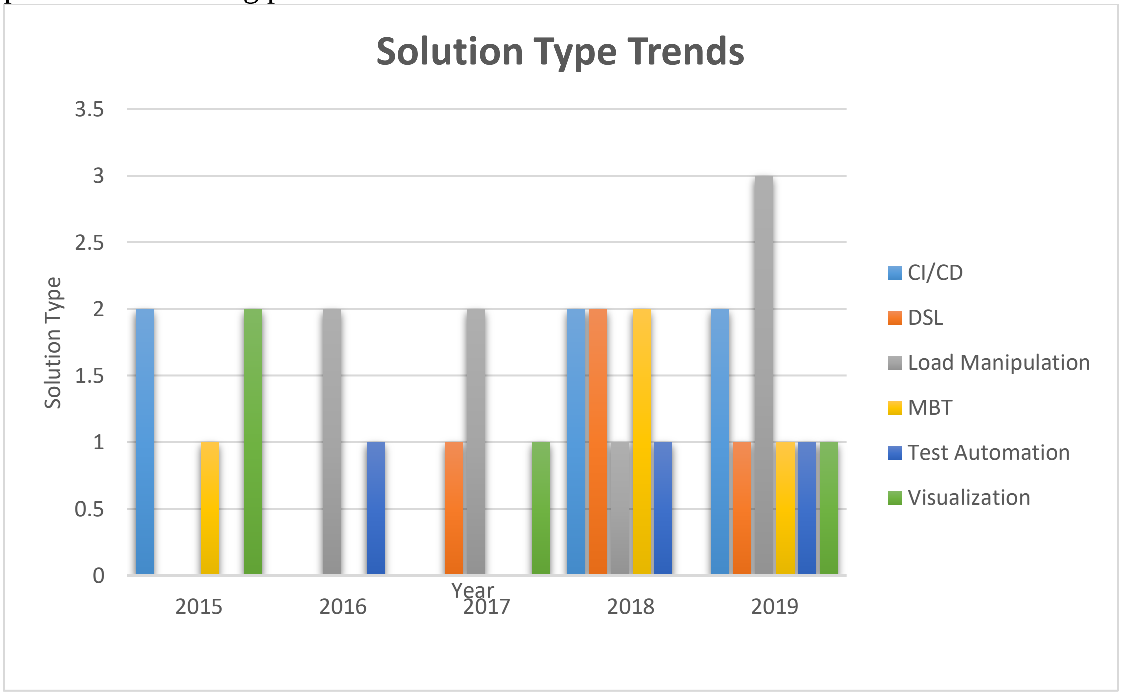

Figure 2.10
Solution Type Trends

Problems Cited

Of the 39 papers analyzed, only 6 of them (7.7%) did not specifically mention problems associated with performance testing or testing in Agile; the rest mention one of several challenges. The following figures visualize the number of papers which site specific challenges, either in performance testing itself or performance testing within Agile software development.

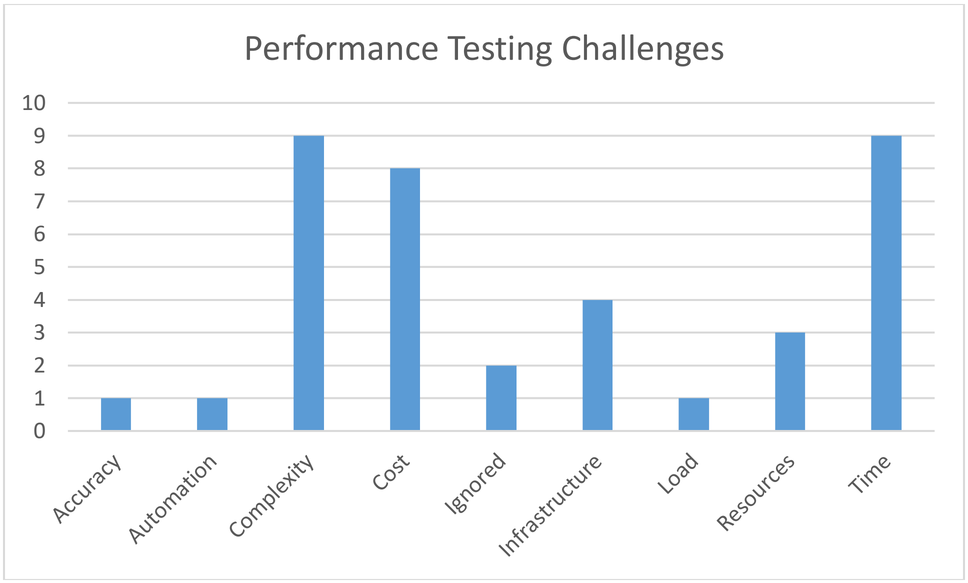

Figure 2.11
Papers citing performance testing challenges

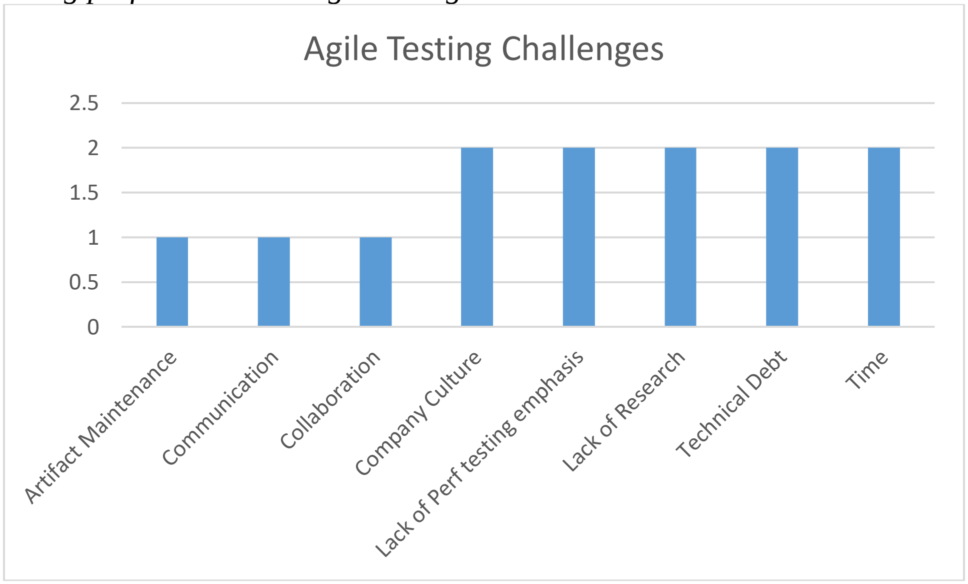

Figure 2.12
Papers citing Agile testing challenges

Complexity and time are tied for the most often cited challenges when it comes to performance testing, followed closely by cost, together making up 68% of the problems noted in the literature. Time is also the challenge most likely to be cited as an Agile challenge when mentioned along with performance testing challenges.

In addition to the count and correlation of performance testing challenges, we can also look at how challenges have been reported over the years. If we remove the challenge types that are only reported once and instead focus on the types of challenges that are reported year over year, we can identify a trend. Figure 11 visualizes this trend and offers some interesting insight. First, despite complexity being among the most often cited challenges in performance testing, its frequency of mention seems to decline over time. Conversely, it appears as though challenges related to cost are becoming increasingly apparent and cited in the research. Not so obvious trends are the decreasing concern of infrastructure challenges but a possible resurgence in concern around cost related challenges in performance testing.

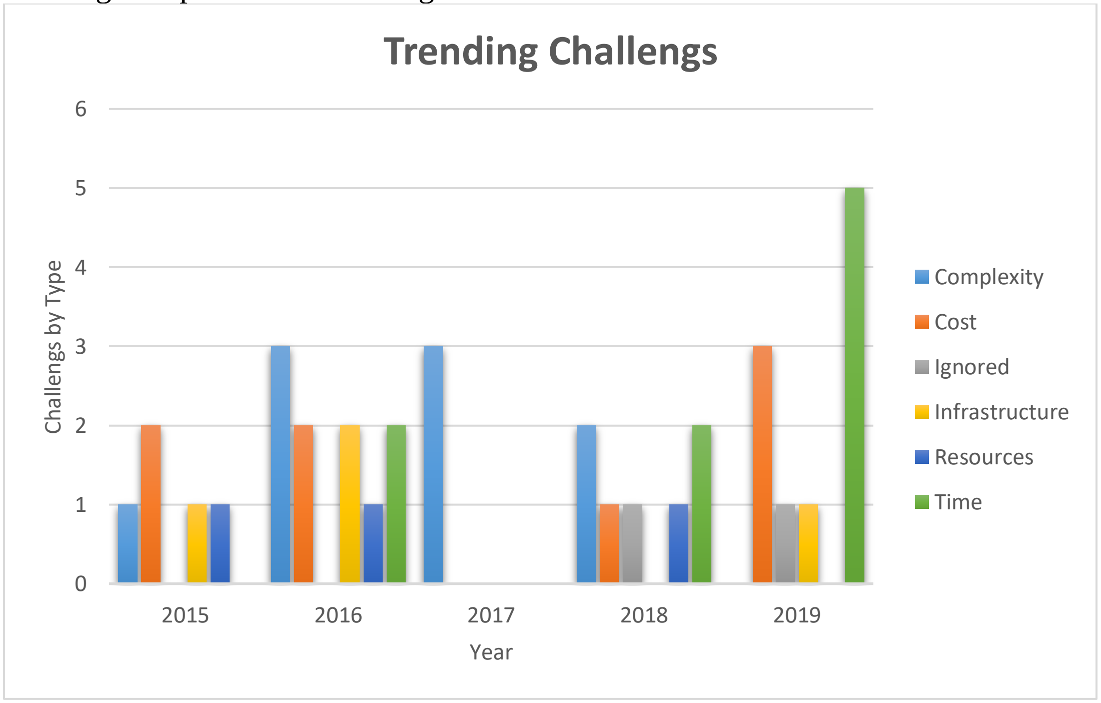

Figure 2.13
Trending Challenges

Qualitative Analysis

In this section, the answers to the research questions posed earlier in the thesis are discussed.

SLR-RQ1. What are the current ideas, challenges, and solutions presented in the peer-reviewed literature regarding performance testing and Agile software development?

There are several takeaways from the analysis of the last five years of research regarding the current state of performance testing and Agile software development. First, as suspected, there is relatively little research and effort in this area, making it difficult to build upon previous efforts. Papers S15 and S28 both specifically note this, while papers S13 and S14 note that teams do not emphasize performance testing enough.

Regarding challenges, the previous sections show that complexity, cost, and time are among the most prevalent challenges in this area. Time specifically is a challenge inherent to the union of performance testing and Agile software development as mentioned in papers S34 and S38 because the release cadence of Agile and the time required for good performance testing are fundamentally at odds.

When it comes to solutions, the research heavily favors load manipulation as a primary solution to problems in performance testing. Additionally, analysis of the last five years of research indicates that domain specific languages (DSLs) are a common companion to other solutions in this space. For example, papers S1, S8, and S38 document the development of a model-based load manipulation tool that is driven by a DSL extending YAML (YAML Ain't Markup Language) scripts.

SLR-RQ2. What trends are emerging in the current peer-reviewed literature regarding performance testing and Agile software development?

With a picture of the current state of performance testing and Agile software development, we can start to look at the data analysis conducted and identify trends in the research. First, it appears as though the problems of performance testing and Agile software development are starting to become apparent to researchers and professionals alike as evidenced in papers S5, S13, S14, S15, and S28. This may imply that research in this area is being recognized as important, or at least as an addressable problem. In any case, it is a contrast to the relative lack of research present in the literature now.

Regarding challenges cited in the research, there does appear to be a trend towards recognizing time as a major constraint. As mentioned in the answer to the previous research question, this is to be expected as Agile software release cadences do not lend themselves to the amount of time needed to conduct performance tests. Additionally, it appears as if challenges related to complexity are being cited less often in recent years. This could mean that complexity in this area is considered a solved problem, or it could mean that it is not the prevalent challenge it was once considered to be.

Trends in solutions seem to point towards a stable popularity in load manipulation and extending CI/CD tools when addressing performance testing challenges. The data does not seem to identify any shift in trends, either negative or positive, but does appear to imply that the types of tools applied to performance testing challenges tend to be done so at random intervals of popularity.

CHAPTER III:
METHODOLOGY

After conducting the literature review and analyzing the data therein, the author endeavored to extend the research by applying the lessons learned. This chapter will describe the design and development of Lulu Performance Test (LPT), the problems it aims to solve, and how it will be tested.

Lulu Performance Test Goals

With the goal of extending the state of performance testing in Agile software development, and after synthesizing the information found in the literature review described in chapter 2, the author believes a performance testing framework would be beneficial to software quality professionals and researchers. To deliver the most benefit to potential users, development of this framework should produce a system with four broad qualities, described here.

Quick Development Time

In the literature review, time was cited as a challenge in both performance testing and Agile development. In many cases, this has to do with the long runtime required for performance testing, but it often refers to the amount of time required to build performance tests as well. Agile methodologies require functionality built in quick iterations and supporting unit and integration tests. These tests, relative to performance and functional tests, are easy to build and thus do not introduce a significant strain on Agile release cadences. As such, to further the field of performance testing in Agile software development, a framework for creating performance tests should model the ease of building unit tests found in the xUnit family of testing frameworks.

Customizability

The literature review reveals that research into performance testing is conducted for various application domains such as mobile, cloud, web, and even embedded systems. Additionally, because different systems have different user bases, the load configurations for proper testing will vary widely from system to system. Finally, different teams will have different testing philosophies relative to their experience and requirements. As such, LPT should be as customizable as possible. In other words, the framework should facilitate user ingenuity in things like gathering metrics and manipulating load configurations rather than defining how these things are done.

Expressive Syntax

Another problem cited in the literature is issues with communication. If possible, the framework should help facilitate conversation and collaboration between testers and development teams. Ideally, the framework should also be able to facilitate better communication between technical and business teams. To achieve this, design and development activities must keep in mind that company and team cultures will be different from one to the next. As such, LPT should facilitate improvements to communication rather than defining or manipulating how communication is conducted.

Compatibility with Continuous Integration/Continuous Delivery Tools

Finally, because the literature often introduces performance testing solutions as CI/CD extensions, it is imperative that the LPT framework be compatible with related tools. Continuous integration, continuous delivery, continuous deployment, version control, and configuration management are all inextricable concepts within Agile software development. For any tool that claiming to improve testing within an Agile context to be successful, it is incumbent upon that tool to plug in easily to these technologies and tools.

Research Questions

The design and development of Lulu Performance Test aims to solve problems around performance testing in Agile software development as mentioned in the literature. Ultimately, this research sets out to answer four research questions related to the performance testing framework to be built:

RQ1. Can it provide a way to rapidly develop performance tests?

RQ2. Can it be customizable enough that users do not have to change their development practices to integrate the framework?

RQ3. Can the framework improve or facilitate communication between development, testing, and business teams?

RQ4. Can the framework be easily integrated into a typical Agile development suite of tools?

Test Protocol

Before development of the LPT framework begins, this study will first define a set of tests the framework will be evaluated against to answer the research questions.

Sandbox

To simulate real-world use, a sandbox was created. The sandbox is a web application that emulates a point of sale (POS) system for a music store; it provides a theoretical system under test (SUT) for which the LPT framework will develop performance tests. The system was built using Python, Flask API, and Postgres SQL, and is deployed to an Amazon Web Services (AWS) Ubuntu Linux image.

The sandbox allows a few very basic customer and employee use cases. These use cases are not important to this study, they merely provide a set of tasks for the performance testing framework to run through during prototyping. Source code for the sandbox system as well build scripts for both local and remote deployment can be found on the author's GitHub. The URL for this source code is https://github.com/erik-whiting/test_site.

Testing for Rapid Development

To evaluate that the system facilitates rapid development of performance tests, the LPT test scripts line count and cyclomatic complexity will be analyzed. The LPT scripts must be compared to other performance testing tools that are text-based. GUI (Graphical User Interface) tools like JMeter, which may be the industry standard, are not comparable to script-based test tools, because lines of code cannot be compared. Additionally, comparing a script or text-based tool to a GUI tool will largely depend on preference. Some people may work faster with a point-and-click workflow while others are more efficient when writing scripts.

Testing for Customization

The goal of customizability for the LPT framework is to minimize disruption to a development team's preexisting workflow. In this stage of the LPT framework development, the author believes the best way to test this is to have the framework utilize test scripts written in different programming languages. If the system can reproduce results when running similar automation scripts in different languages, it can be concluded that the system is customizable.

Testing for Continuous Integration/Continuous Delivery Integration

As mentioned in a previous section, it is imperative that the LPT framework be easily implemented into a CI/CD system, and that it will work well with typical tools used in Agile software development. Therefore, to test the LPT framework's ability to plug into these tools, the following test is defined:

The sandbox system will be given a tests directory. First, this test directory will include a few basic unit tests to emulate what a production level project might have. In addition to these unit tests will also be LPT scripts. The author will then link the repository to a TravisCI project via a GitHub hook. This will run all the tests for the sandbox every time new commits are pushed to the repository. If the TravisCI build runs as expected, it can be concluded that the LPT framework has the ability to plug into CI/CD systems and other Agile software development tools.

Sandbox Design

For the sake of completeness, this section will briefly describe the web application sandbox, and how it is used for this research. The sandbox application simulates a music store in which customers can buy records and employees can manage sales. The application is built using Flask API, a Python framework, and deployed to Amazon Web Services (AWS) and thus has a dynamically generated URL.

The application architecture is fairly simple. There is an object relational mapper (ORM) module with a Connection and Query class. These classes abstract database connection functions and query results parsing. There is also a Resources module, which implements the ORM module. The Resources module handles specific operations to be conducted on the various data models of the application. Finally, there is the rest of the model-view-controller (MVC) architecture which defines the rest of the application.

There are three main use cases for the sandbox application. The first is general browsing from the customer’s point of view. This includes clicking through bands and their albums and seeing song lists of those albums. The second use case is also from the customer’s perspective and simulates an online shopping experience. Customers can select form the store inventory, see their bill, and place an order. The final use case is from an employee point of view and involves looking through sales records. This is a basic view of how much of what items were sold and when they were sold.

Lulu Performance Test Design and Architecture

Lulu Performance Test (LPT) aims to solve two of the main problems cited in the literature—time and communication—by utilizing three of the solution types cited by the literature: load manipulation, domain specific language, and CI/CD extensions. The LPT framework aims to improve the communication between test teams, development teams, and business teams by providing a configurable DSL based on concepts common to all software systems.

On a technical level, the LPT framework provides a means for defining metrics of a system to be measured, as well as a means for measuring common system metrics such as memory usage, CPU usage, and disk space usage. The system allows test engineers to write scripts that poll these metrics on configurable time intervals. The system polls and returns these metrics as decimal numbers, allowing script developers to use these results however their teams prefer.

On a business level, the LPT framework assigns use cases to user profiles, which comprise user groups. The framework also allows use case automation scripts to be written in any language as long as the scripts can be run from a command line. This means that test engineers can write a test script, assign it to a user profile (e.g., “Bob”), assign that user profile to a user group, (e.g., “Accounting”) and then use these to define a typical workload. The high-level architecture of the system is shown below.

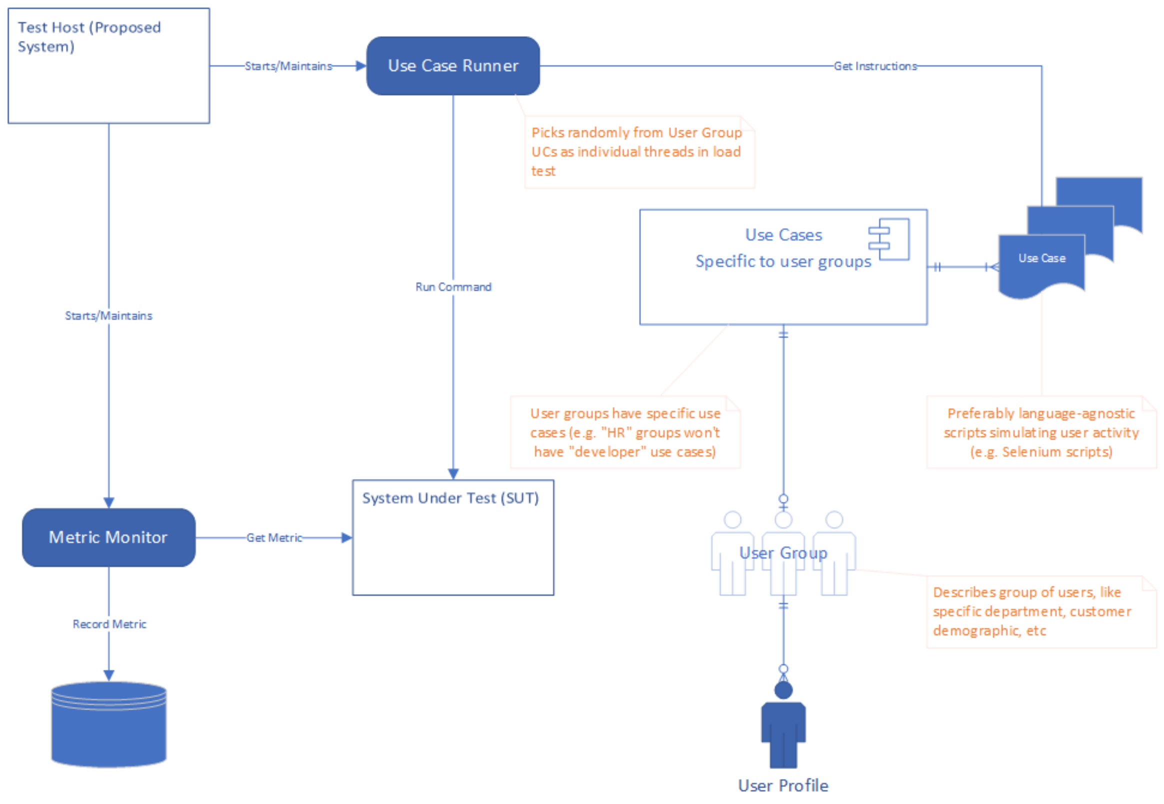
Test Host (Proposed System)
Starts/Maintains
Use Case Runner
Get Instructions
Picks randomly from User Group UCs as individual threads in load test
Use Cases Specific to user groups
Use Case
Run Command
Starts/Maintains
User groups have specific use cases (e.g. "HR" groups won't have "developer" use cases)
Preferably language-agnostic scripts simulating user activity (e.g. Selenium scripts)
System Under Test (SUT)
Metric Monitor
Get Metric
User Group
Describes group of users, like specific department, customer demographic, etc
Record Metric
User Profile

Figure 3.1
LPT Framework High-Level architecture

LPT monitors system processes via threading, where each item to be monitored is done so on a single logical thread. The Monitors module features a class called MetricMonitor which implements two interfaces. The first interface is called Monitoring and is defined in LPT. The second is called Runnable and is a built-in Java library for multithreaded applications. MetricMonitor is the supertype for four subtypes: MemoryMonitor, CpuMonitor, DiskSpaceMonitor, and OtherMonitor. The first three provide out-of-the-box capability for monitoring the metric they are named after. The OtherMonitor class provides developers with a way to implement LPT’s Monitors class with a system level command for monitoring a metric the developer may want to monitor. The Monitors module is visualized in the figure below:

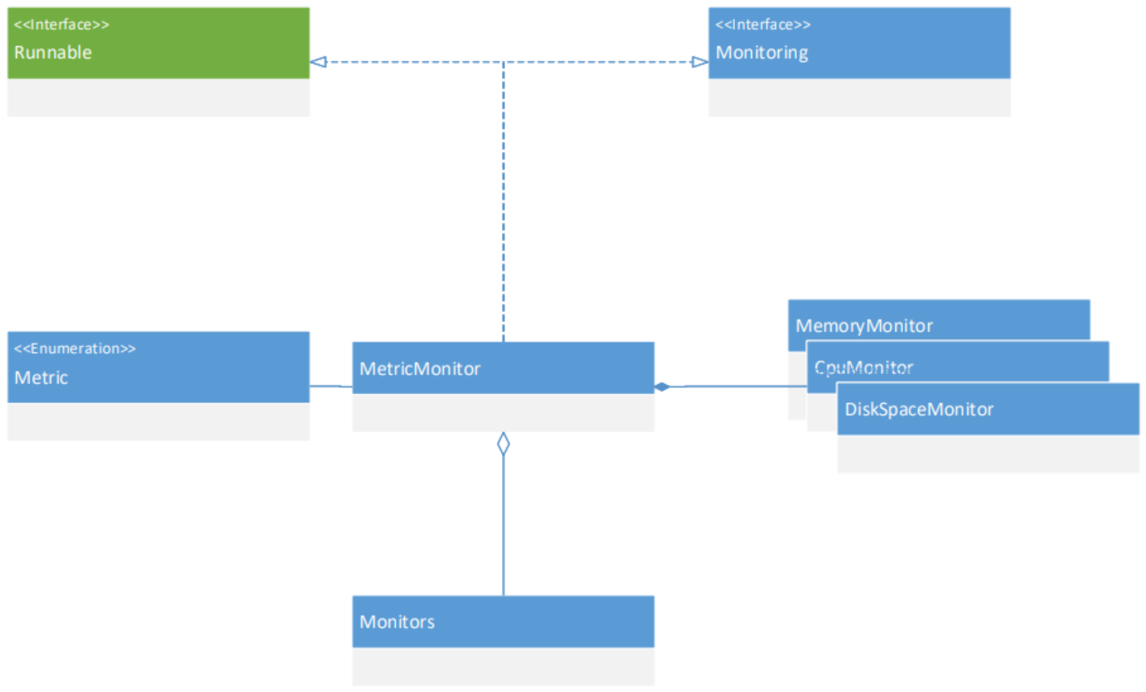

Figure 3.2
Monitors module

Additionally, the class diagram generated by the development IDE (JetBrains) provides a more technical illustration of the module:

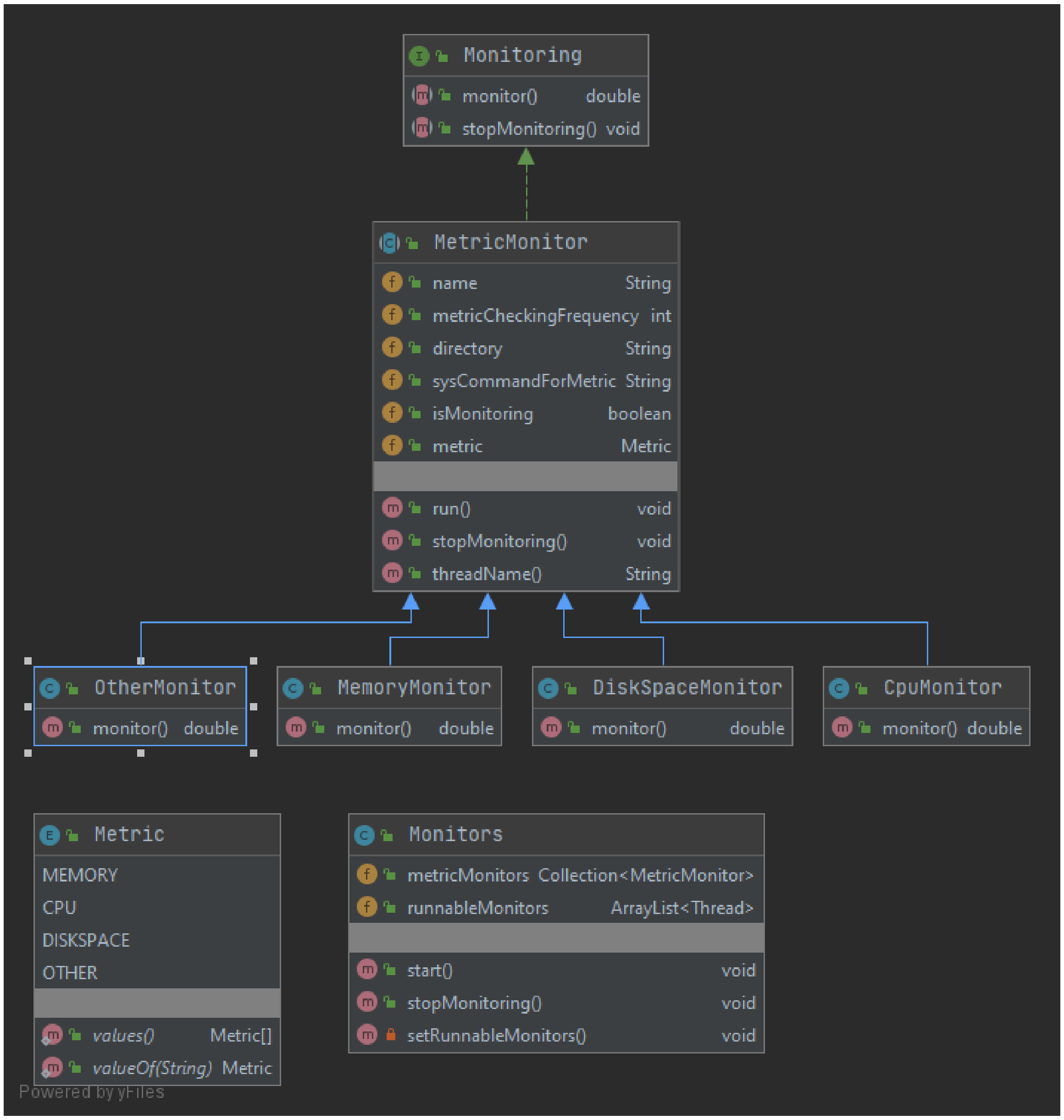

Figure 3.3
Monitors module class diagram

Running in parallel with the monitors are the actual test scripts. Users define the location of these automation scripts which the user also defines. The framework will run the scripts while also running the metric monitors, giving users an idea of system performance under any kind of load the user defines.

CHAPTER IV:
RESULTS

Results of the testing performed will be described here. Lulu Performance Test (LPT) had four goals and four research questions.

Rapid Development Test Results

One of the main goals of LPT was to provide testers and developers in an Agile software development environment an efficient way to write performance tests. To evaluate LPT's efficacy in rapid development, we wrote a test script using the LPT Domain Specific Language (DSL) and evaluate it against a similar JMeter Script. Originally, the cyclomatic complexity of the DSL script was to be evaluated as well, but because the DSL is written in JavaScript Object Notation (JSON), it is not possible to calculate cyclomatic complexity (or the cyclomatic complexity is calculated as 0).

The following script, written in the LPT DSL runs two test scripts, each with two threads, and monitors the memory and CPU usage of the system under test (SUT):

```
{
  "Performance Test": {
    "name": "default",
    "useCases": [
      {
        "name": "Explore Albums",
        "script":
"C:\\Users\\eedee\\Documents\\test_site_tests\\customer_user_group\\exp
lore_albums.py",
        "command": "python",
        "threads": "2"
      },
      {
        "name": "Explore Bands",
        "script":
"C:\\Users\\eedee\\Documents\\test_site_tests\\customer_user_group\\exp
lore_bands.py",
        "command": "python",
        "threads": "2"
      }
    ],
    "monitors": [
      {
        "name": "memory", "every": "500"
      },
      {
        "name": "CPU", "every": "500"
      }
    ]
  }
}
```
Figure 4.1
LPT DSL Script

The script above is 27 lines when formatted for easy reading. Functionally, only 14 of the lines are necessary and contribute to the test script (the rest are bracket-or-comma-only lines). To build a similar script, the developer need only know the location of a test script, the command needed to run that script, and to know which metrics to monitor.

This test script will be compared against a similar test script written in Python. Python is chosen because it tends to be less verbose than languages like Java and because it is a common tool to use in test automation. The following script is a Python script that runs a similar test and monitor combination, but only once:

```python
import threading
import psutil

def monitorMemory():
    proc = psutil.Process()
    print(proc.memory_info().rss)

def monitorCPU():
    print(psutil.cpu_percent())

def runTestScript(path_to_script):
    # test script running code
    print("running")

if __name__ == "__main__":
    thread1 = threading.Thread(target=monitorMemory)
    thread2 = threading.Thread(target=monitorCPU)
    thread3 = threading.Thread(target=runTestScript, args=("path1",))
    thread3 = threading.Thread(target=runTestScript, args=("path2",))
    threads = [thread1, thread2, thread3, thread4]
    for thread in threads:
        thread.start()
```

Figure 4.2
Python Performance Test

The script above is 22 lines, 17 of which are not whitespace. This script requires more lines of code than the LPT domain specific language (DSL) to run a single instance of the performance test to be conducted, in order to run multiple iterations like the LPT script does, even more lines would have to be added. From this information, it can be concluded that the LPT DSL facilitates comparatively rapid performance test development.

Customization Test Results

The goal of customization in the context of LPT is to provide developers a technology-agnostic tool for running performance tests. LPT is designed to run automation scripts on a system while monitoring that system's metrics. In the development phase, Python automation scripts were used against the Sandbox system. To test the customizability of LPT, a similar test script in a different language will be developed and used as input to the LPT DSL script. Here is a test script for exploring the bands page in the sandbox (note the test script uses a library called LuluTest, a browser automation framework developed by the author):

```python
import random
from LuluTest import *

from vars import Vars
from customer_profile import CustomerProfile

customer = CustomerProfile()

vars = Vars()
page = vars.new_page()
actions = vars.new_actions() # Make headless by passing False

actions.go(page)
customer.linger()
actions.click(
   PageElement(('id', 'bands'), 'bands')
)
customer.linger()

band_id = random.randrange(30)
actions.click(
   PageElement(('id', f'band-{band_id}'), 'random band')
)
customer.get_distracted()

actions.click(
   PageElement(('link text', 'Bands'), 'back')
)
customer.linger()
actions.close()
```

Figure 4.3
Python automation script

A similar script, albeit one without the simulated wait times, in Ruby is defined as follows:

```ruby
require 'watir'

browser = Watir::Browser.new
browser.goto('127.0.0.1:5000')
browser.link(text: 'Albums').click

rand_album = rand(10)
browser.goto("127.0.0.1:5000/bands/#{rand_album}")

browser.link(text: 'Bands').click
rand_album = rand(20)
browser.goto("127.0.0.1:5000/bands/#{rand_album}")
browser.close
```

Figure 4.4
Ruby automation script

To test that both of these scripts can be run by LPT, we define the following prototype in Java and observe the results:

```java
package com.lulu.main.prototype;

import com.lulu.main.java.models.use_cases.UseCase;
import com.lulu.main.java.models.use_cases.UseCases;

import java.util.ArrayList;

public class UseCasePrototype {
    public static void main(String[] args) throws InterruptedException {
        String ucCustomerScriptRoot = "C:\\Users\\eedee\\Documents\\test_site_tests\\customer_user_group";
        String customerUcScript1 = ucCustomerScriptRoot + "\\explore_albums.py";
        String customerUcScript2 = ucCustomerScriptRoot + "\\explore_albums.rb";

        UseCases useCases = new UseCases(new ArrayList<>() {
            {
                add(new UseCase("Explore Albums (python)", customerUcScript1, "python", 2));
                add(new UseCase("Explore Albums (ruby)", customerUcScript2, "ruby", 2));
            }
        });

        useCases.start();
        System.out.println(useCases.doneRunning());
    }
}
```

Figure 4.5
Language-agnostic automation

Running this script works as expected, both the Python and Ruby script run the commands against a browser. As long as a system command and script can be passed to the UseCase class, LPT can run the script. This proves that LPT can be customized and run whatever technologies the users want to use.

Continuous Integration and Continuous Delivery Pluggability Test Results

Most importantly, the framework’s ability to plug into Agile tools such as version control and Continuous Integration/Continuous Delivery (CI/CD) tools is tested. To complete this test within the time allotted, the author implemented some non-standard techniques to implement Lulu Performance Test into these tools.

Lulu Performance Test (LPT) is first compiled into a Java Archive (or JAR) file and moved into the Test Site tests directory. Within this directory is also a similar LPT script as the JavaScript Object Notation (JSON) file detailed in figure 4.1. Essentially, to run the performance test, the application must be started, and the JAR file ran with the DSL file location as input. To accomplish this, the Test Site application's repository is hosted in GitHub and configured to run a build script in TravisCI every time a new commit is pushed to the repository.

To accomplish this testing strategy, and to emulate what a typical build for an application like Test Site might look like, the following build script drives the TravisCI integration and testing:

```
1. language: python
2. python:
3.   - "3.6"
4.
5. services:
6.   postgresql
7.
8. addons:
9.   chrome: stable
10.
11.  after_script:
12.    - chmod +x ~/build.sh
13.    - chmod 777 ./tests/LPT/LuluPerfTest.jar
14.    - bash ~/build.sh
15.    - sudo java -jar ./tests/LPT/LuluPerfTest.jar
   ./tests/LPT/basic_test.json
16.
17.  script: python -m unittest discover tests
18.
19.  install:
20.    - pip install flask
21.    - pip install psycopg2
22.    - wget -N
   https://chromedriver.storage.googleapis.com/83.0.4103.39/chromedr
   iver_linux64.zip -P ~/
23.    - unzip ~/chromedriver_linux64.zip -d ~/
24.    - rm ~/chromedriver_linux64.zip
25.    - sudo mv -f ~/chromedriver /usr/local/share/
26.    - sudo chmod +x /usr/local/share/chromedriver
27.    - sudo ln -s /usr/local/share/chromedriver
   /usr/local/bin/chromedriver
28.    - sudo apt-get install binfmt-support
```

Figure 4.5
Travis CI Build script (.travis.yml)

Note lines 11 through 16. These lines first change permissions of both the build script and JAR file to be executable. The next portion runs the build script located in the Test Site repository so that the application is actually running. Finally, the Lulu Performance Test JAR is invoked with the DSL script as input. When run in the TravisCI build server, the LPT script runs as expected. This proves that the LPT framework is pluggable into Agile tools.

Not Tested: Expressive Syntax

One of the goals of LPT is to provide an expressive syntax via the DSL to facilitate communication between test teams, development teams, and business teams. The time and scope of this thesis project did not allow for testing this goal. In order to properly test the DSL expressiveness, a survey would have to be conducted among professionals within the aforementioned functional units. The survey would feature scripts from a more fully defined DSL and ask respondents what they meant. The more correct answers on the survey would mean the syntax was expressive and a good tool for facilitating communication among these disparate teams. As mentioned before, the time required to complete a survey and analyze the data is not conducive to the scope and timeline of this project.

Answers to Research Questions

The rest of this chapter will elaborate on the answers to the study's research questions

RQ1. Can Lulu Performance Test provide a way to rapidly develop performance tests?

Yes, the DSL provided by Lulu Performance Test (LPT) can generate performance tests qin fewer lines and less complex scripts than general purpose programming languages. LPT was not compared against tools like JMeter or Postman, since these tools utilize a graphical user interface and cannot be easily compared with a framework using a DSL to script tests.

RQ2. Can Lulu Performance Test be Customizable Enough that Users do not Have to Change Their Development Practices to Integrate the Framework?

Yes, this study proved the LPT will work when given either a Python or Ruby script. The framework works by utilizing system commands to run automation scripts. As long as an automation script can be invoked from the command line, it can be used as an automation script with LPT.

RQ3. Can Lulu Performance Test Improve or Facilitate Communication Between Development, Testing, and Business Teams?

This research question remains unanswered. There were no tests conducted to evaluate if LPT's domain specific language would facilitate or improve communication between development, testing, and business teams.

RQ4. Can Lulu Performance Test be Integrated into a Typical Agile Development Suite of Tools?

Yes, although its current implementation in the test application is not scalable. The pluggability of LPT has been proven in this study, however this pluggability can be implemented in more efficient ways. Further work is needed in this area.

CHAPTER V:

CONCLUSION AND FUTURE WORK

Literature Review Lessons Learned

Analysis of the literature as described in the literature review revealed current states and apparent trends. In addition, the analysis also allows for further questions to be asked and opens some paths to further research.

First, the decline of complexity coupled with the rise of cost and time in performance testing challenges seems to be an anomaly. The author hypothesizes that this may be due to the increasing popularity of cloud-based application development. It is hypothesized as such due to the componentization of different pieces of infrastructure (i.e., database servers, virtualized networks, and so on) that allow for a higher level of atomicity when troubleshooting performance bottlenecks. This could also be due to performance monitoring capabilities in the cloud, a subject which—upon a cursory literature database search—appears to already have a sizeable body of work. It is likely worth researching the relationship between reduced complexity in performance testing and the proliferation of cloud-based applications.

As noted in the section describing solutions, one paper (S23) introduces the idea of fault injection which draws ideas from Netflix's Chaos Monkey approach to testing [24] [25]. This type of solution falls into its own category due to its uniqueness. However, it is possible a more focused approach to collecting data in this area would be beneficial to understanding what value "Chaos Engineering" can or does bring to performance testing. This area of research also seems to be growing in interest judging from a quick search in the IEEE Xplore and ACM Digital Libraries.

Future Work

This study aimed to address some of the challenges present in the union of performance testing and Agile software development. The literature review component of the study identifies the current state and emerging trends of performance testing. The application developed in tandem with this study shows that ideas from the literature can be consolidated into a single tool. There is a lot of research left undone in this study, however, and performance testing is still at odds with Agile software development. There are two areas in which further research may benefit the software engineering community.

Further Research into Chaos Testing

As mentioned before, the area of chaos testing—and the broader discipline of chaos engineering—were only barely touched on in this study. The author believes that this area shows promise in both solving unique problems as well as yielding interesting findings. If given more time, a similar literature review of these areas could be conducted to provide a starting point for future research.

Improvements to the Lulu Performance Test Framework

Chapter IV mentions that one of the Lulu Performance Test (LPT) framework goals was to facilitate communication between teams of differing disciplines. The timeline of this study did not lend itself to looking into the realization of this goal. To further develop and analyze LPT's efficacy in improving communication, there are a few things that must be done. First, the domain specific language (DSL) should be ported to its own standalone language rather than being written in JSON. Second, scripts of this independent DSL should be developed and presented to LPT's intended users. From here, the users would say whether or not the scripts accurately describe what they are doing. Responses would then be analyzed and updated according to respondent feedback leading to incremental improvements to the DSL and, eventually, an effective communication and technical tool.

Additionally, LPT's system reporting features need improvement. Currently, the monitoring classes report metrics via standard output in the command line; a more appropriate way to report results would be either by recording data in a database or some kind of spreadsheet software. In this way, performance test results could be persisted and analyzed by teams for better quality improvements. Furthermore, reporting dashboards could be developed and pass/fail configurations could be defined to aid in continuous deployment environments.

Finally, LPT delivery does not use best practices. In this study, a JAR file containing the LPT files are stored in a system's test folder and called on the command line via a build script. The more appropriate way to utilize a tool like this would be to clone or remotely download the LPT test runner classes, compile in the build server, and then ran with user-provided DSL scripts. This would allow teams to customize their build environments with the Java versions they prefer as well as allow users to get an updated and centralized version of LPT rather than a JAR file the author compiled on his personal computer, then copy and pasted into a test directory.

Unifying Rapid Test Development Tools

The automation scripts used in this study implement an open source automation framework developed by the author known as LuluTest (https://github.com/erik-whiting/LuluTest). The author's goals with LuluTest were much like the goals of Lulu Performance Test: to provide Agile development teams a tool to develop tests quickly. LuluTest also has the added goal of allows tests to be robust and easy to understand. Agile development relies on exhaustive testing, and thus improving the efficiency and cost effectiveness of Agile testing is always a good contribution to software engineering.

Conclusion

To conclude, the benefits of Agile software development are due in no small part to its philosophy regarding testing. Generally speaking, better tests means better products. However, there are several challenges in the realm of performance or non-functional requirements testing. The time required to conduct proper performance testing is fundamentally at odds with the release cadence of Agile software development. Many kinds of solutions to this problem exist, and it is possible to unify these approaches.

REFERENCES

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, *Manifesto for Agile Software Development*, 2001.

[2] H. Guang-yong, "Study and practice of import Scrum agile software development," in *2011 IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, 2011.

[3] J. a. M. R. C. Newkirk, "Extreme Programming in Practice," in *Addendum to the 2000 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum)*, Minneapolis, 2000.

[4] U. W. M. Shafiq, "Documentation in Agile Development A Comparative Analysis," in *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, Karachi, 2018.

[5] R. A. DeMillo, "Software Testing," in *Encyclopedia of Computer Science*, John Wiley and Sons Ltd., 2003, p. 1645–1649.

[6] E. F. d. L. V. F. Collins, "Software Test Automation Practices in Agile Development Environment: An Industry Experience Report," in *Proceedings of the 7th International Workshop on Automation of Software Test*, Zurich, 2012.

[7] R. Khan, A. K. Srivastava and D. Pandey, "Agile approach for Software Testing process," in *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*, Moradabad, 2016.

[8] A. Freitas and R. Vieira, "An Ontology for Guiding Performance Testing," in *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web*

Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 01, Warsaw, 2014.

[9] Z. M. J. Jiang, "Load Testing Large-Scale Software Systems," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Florence, 2015.

[10] S. Pradeep and Y. K. Sharma, "A Pragmatic Evaluation of Stress and Performance Testing Technologies for Web Based Applications," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, Dubai, 2019.

[11] A. Mehta, J. Dürango, J. Tordsson and E. Elmroth, "Online Spike Detection in Cloud Workloads," in *2015 IEEE International Conference on Cloud Engineering*, Tempe, AZ, 2015.

[12] S. V. E. Tchagou, A. Termier, J.-F. Mehaut, B. Videau, M. Santana and R. Quiniou, "Reducing Trace Size in Multimedia Applications Endurance Tests," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, Grenoble, 2015.

[13] O. Sinanoglu and E. J. Marinissen, "Analysis of the Test Data Volume Reduction Benefit of Modular SOC Testing," in *Proceedings of the Conference on Design, Automation and Test in Europe*, Munich, 2008.

[14] P. Moura and F. Kon, "Automated Scalability Testing of Software as a Service," in *Proceedings of the 8th International Workshop on Automation of Software Test*, San Francisco, 2013.

[15] L. Crispin and J. Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley Professional, 2009.

[16] V. Ferme and C. Pautasso, "A Declarative Approach for Performance Tests Execution in Continuous Software Development Environments," in *2018*

ACM/SPEC International Conference on Performance Engineering (ICPE '18), New York, 2018.

[17] A. O. Portillo-Dominguez, M. Wang, J. Murphy, D. Magoni, N. Mitchell, P. F. Sweeney and E. Altman, "Towards an automated approach to use expert systems in the performance testing of distributed systems," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, San Jose, 2014.

[18] H. Schulz, D. Okanovic, A. van Hoorn, V. Ferme and C. Pautasso, "Behavior-Driven Load Testing Using Contextual Knowledge - Approach and Experiences," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, Mumbai, 2019.

[19] A. Kattepur and M. Nambiar, "Service Demand Modeling and Prediction with Single-User Performance Tests," in *Proceedings of the 9th Annual ACM India Conference*, Gandhinagar, 2016.

[20] T. Field, R. Chatley and D. Wei, "Software Performance Testing in Virtual Time," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, Berlin, 2018.

[21] V. R, B. Silva Estacio, R. Prikladnicki and T. Greechenig, "Systematic Literature Review on Agile Practices in Global Software Development," *Information and Software Technology*, vol. 96, pp. 161-180, 2018.

[22] F. Abbors and D. Truscan, "Approaching Performance Testing from a Model-Based Testing Perspective," in *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, Nice, France, 2010.

[23] M. Utting and B. Legeard, Practical model-based testing: A Tools Approach, Elsevier Science, 2010.

[24] M. A. Chang, B. Tschaen, T. Benson and L. Vanbever, "Chaos Monkey: Increasing SDN Reliability through Systematic Network Destruction," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, London, 2015.

[25] n. T. Blog, "The Netflix Simian Army," 19 July 2011. [Online]. Available: https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116. [Accessed 21 Austust 2020].

[26] A. C. Z. S. H. A. S. P. F. M. T. D. Hellmann, "Agile Testing: A Systematic Mapping across Three Conferences: Understanding Agile Testing in the XP/Agile Universe, Agile, and XP Conferences," in *2013 Agile Conference*, Nashville, 2013.

APPENDIX A:

INCLUDED STUDIES

Article ID	**Article Name**	**Author(s)**	**Year**	**Database**
S1	A Declarative Approach for Performance Tests Execution in Continuous Software Development Environments	Ferme, Vincenzo; Pautasso, Cesare	2018	ACM
S2	A Systematic Review on Cloud Testing	Bertolino, Antonia; Angelis, Guglielmo De; Gallego, Micael; Garcia, Boni; Gortazar, Francisco; Lonetti, Francesca; Marchetti, Eda	2019	ACM
S3	A Systematic Review on the Use of Definition of Done on Agile Software Development Projects	Silva, Ana; Araujo, Thalles; Nunes, Joao; Perkusich, Mirko; Dilorenzo, Ednaldo; Almeida, Hyggo; Perkusich, Angelo	2017	ACM

S4	Adopting Continuous Integration and Continuous Delivery for Small Teams	M. K. A. Abbass; R. I. E. Osman; A. M. H. Mohammed; M. W. A. Alshaikh	2019	IEEE
S5	Agile Team Members Perceptions on Non-functional Testing: Influencing Factors from an Empirical Study	C. R. Camacho; S. Marczak; D. S. Cruzes	2016	IEEE
S6	An analysis of automated tests for mobile Android applications	D. Bernardo Silva; A. T. Endo; M. M. Eler; V. H. S. Durelli	2016	IEEE
S7	An Architecture to Automate Performance Tests on Microservices	de Camargo, Andre; Salvadori, Ivan; Mello, Ronaldo dos Santos; Siqueira, Frank	2016	ACM
S8	Behavior-Driven Load Testing Using Contextual Knowledge - Approach and Experiences	Schulz, Henning; Okanovic, Duvsan; van Hoorn, Andre; Ferme, Vincenzo; Pautasso, Cesare	2019	ACM
S9	Browser Performance of JavaScript Framework, SAPUI5 & jQuery	J. Raigoza; R. Thakkar	2016	IEEE

S10	Challenges in Adapting Agile Testing in a Legacy Product	R. K. Gupta; P. Manikreddy; A. GV	2016	IEEE
S11	Challenges in Assessing Technical Debt Based on Dynamic Runtime Data	M. Ciolkowski; L. GuzmÃ¡n; A. Trendowicz; A. M. Vollmer	2018	IEEE
S12	Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management	S. A. I. B. S. Arachchi; I. Perera	2018	IEEE
S13	Continuous Performance Testing in Virtual Time	R. Chatley; T. Field; D. Wei	2019	IEEE
S14	Continuous Software Testing in a Globally Distributed Project	N. B. Moe; D. Cruzes; T. DybÃ¥; E. Mikkelsen	2015	IEEE
S15	Continuous Testing in the Development of IoT Applications	L. G. GuÅŸeilÄƒ; D. Bratu; S. Moraru	2019	IEEE

S16	Crowd and Laboratory Testing, Can They Co-exist? An Exploratory Study	F. Guaiani; H. Muccini	2015	IEEE
S17	Empowering Dynamic Task-Based Applications with Agile Virtual Infrastructure Programmability	H. Zhou; Y. Hu; J. Su; M. Chi; C. de Laat; Z. Zhao	2018	IEEE
S18	Enabling Performance Management During Cloud Applications Migration	N. Chawla; D. Kumar; D. Sharma	2019	IEEE
S19	FIG based Quality Assurance in Software Product Lines	N. Yousaf; R. Sheikh; M. Abbas	2017	IEEE
S20	Initial Experiments with Duet Benchmarking: Performance Testing Interference in the Cloud	L. Bulej; V. HorkÃ; P. TÅ¯ma	2019	IEEE
S21	Model-Based Performance Evaluations in Continuous Delivery Pipelines	Dlugi, Markus; Brunnert, Andreas; Krcmar, Helmut	2015	ACM

S22	Navigate, Understand, communicate: How Developers Locate Performance Bugs	S. Baltes; O. Moseler; F. Beck; S. Diehl	2015	IEEE
S23	Online Robustness Testing of Distributed Embedded Systems: An Industrial Approach	K. Alnawasreh; P. Pelliccione; Z. Hao; M. RÃ¥nge; A. Bertolino	2017	IEEE
S24	PerfVis: Pervasive Visualization in Immersive Augmented Reality for Performance Awareness	Merino, Leonel; Hess, Mario; Bergel, Alexandre; Nierstrasz, Oscar; Weiskopf, Daniel	2019	ACM
S25	Poster: ClearTH Test Automation Framework: A Running Example of a DLT-Based Post-Trade System	V. Panarin; A. Bulda; I. Itkin; A. Zverev; K. Zagorouiko; M. Mamedov; A. Rybakova; A. Gromova; E. Treshcheva; S. Tishin; R. Yavorskiy	2019	IEEE

S26	Practices to Make Agile Test Teams Effective: Challenges and Solutions	T. Anand; V. S. Mani	2015	IEEE
S27	Pragmatic Scrum Transformation: Challenges, Practices & Impacts During the Journey A Case Study in a Multi-Location Legacy Software Product Development Team	Gupta, Rajeev Kumar; Manikreddy, Prabhulinga; Arya, K. C.	2017	ACM
S28	Quality Assurance Practices in Continuous Delivery - an implementation in Big Data Domain	A. Cheriyan; R. R. Gondkar; T. Gopal; S. B. S.	2018	IEEE
S29	Requirements Engineering and Software Testing in Agile Methodologies: A Systematic Mapping	Coutinho, Jarbele C. S.; Andrade, Wilkerson L.; Machado, Patricia D. L.	2019	ACM
S30	Service Demand Modeling and Prediction with Single-User Performance Tests	Kattepur, Ajay; Nambiar, Manoj	2016	ACM
S31	Setting Realistic Think Times in Performance	Ramakrishnan, Raghu; Shrawan,	2017	ACM

	Testing: A Practitionerâ€™s Approach	Vandana; Singh, Prabhpahul		
S32	Social dogfood: A framework to minimise cloud field defects through crowd sourced testing	D. Malone; J. Dunne	2017	IEEE
S33	Software Crowdsourcing Practices and Research Directions	E. Bari; M. Johnston; W. Wu; W. Tsai	2016	IEEE
S34	Software Performance Testing in Virtual Time	Field, Tony; Chatley, Robert; Wei, David	2018	ACM
S35	Survey of Testing Methods of O2O Catering Platform	Xu, Hefang; Su, Caihong; Wu, Shaoyu; Tang, Dongping	2019	ACM
S36	Test orchestration a framework for Continuous Integration and Continuous deployment	N. Rathod; A. Surve	2015	IEEE
S37	The Internet of (Showbiz) Things: Scalability Issues in Deploying and Supporting	Ledwidge, Michela; Burrell, Dr Andrew	2015	ACM

	Networked Multimedia Experience.			
S38	Towards Holistic Continuous Software Performance Assessment	Ferme, Vincenzo; Pautasso, Cesare	2017	ACM
S39	VICTORY: A New Approach to Automated Vehicle Testing	T. Thompson; K. Saylor	2018	IEEE

APPENDIX B:

PAPER CATEGORIZATION INFORMATION

Id	**Paper Type**	**Perf. Challenges**	**Agile Challenges**	**App Domain**	**Solution Type**
S1	Solution	Complexity; Cost			DSL; MBT
S2	Lit Review	Cost; Time		Cloud	
S3	Lit Review		Technical Debt		
S4	Solution	Time			CI/CD
S5	Report	Complexity; Cost; Time; Awareness	Company Culture		
S6	Report	Resources		Mobile	
S7	Solution	Infrastructure		Microservices	Test Automation; Load Manipulation
S8	Solution	Automation			DSL; Load Manipulation
S9	Report			Web	
S10	Report	Complexity			
S11	Report	Complexity			

S12	Solution	Load			Load Manipulation; CI/CD
S13	Solution	Cost; Time	Lack of Perf testing emphasis		Load Manipulation; MBT
S14	Report		Lack of Perf testing emphasis		
S15	Solution		Lack of Research	IoT	CI/CD
S16	Report	Cost			
S17	Solution	Resources			Test Automation; DSL
S18	Report	Time		Cloud	
S19	Solution	Complexity		SPL	Visualization
S20	Solution	Cost; Time; Infrastructure		Cloud	Load Manipulation
S21	Solution	Resources; Infrastructure			MBT; CI/CD
S22	Report	Complexity			
S23	Solution	Complexity		Embedded	Fault Injection
S24	Solution				Visualization
S25	Solution			Blockchain	Test Automation

S26	Report				
S27	Report		Communication; Collaboration; Technical Debt		
S28	Solution		Lack of Research; Company Culture	Big Data	CI/CD
S29	Lit Review		Artifact Maintenance		
S30	Solution	Time; Infrastructure			Load Manipulation
S31	Solution	Accuracy			Load Manipulation
S32	Report			Cloud	
S33	Report	Complexity; Cost			
S34	Solution	Time; Ignored	Time		MBT
S35	Report	Ignored		Web	
S36	Solution	Cost			CI/CD; Visualization
S37	Solution			IoT	Visualization
S38	Solution	Complexity	Time		DSL; Load Manipulation
S39	Solution	Time			

APPENDIX C:
REMOVED STUDIES

Article Name	**Authors**	**Year**	**Phase Removed**
Fast and accurate synthesis of electronically reconfigurable annular ring monopole antennas using particle swarm optimisation and artificial bee colony algorithms	E. J. Brito Rodrigues; H. W. Castro Lins; A. G. D'AssunÃ§Ã£o	2016	Title Review
Laguna Patroller: Mobile Application for Public Awareness about Violence with Global Positioning System and Image Processing	M. F. S. Algaba; R. A. Peji; A. L. S. Maria; J. M. Bawica	2018	Title Review
A temperature monitoring system incorporating an array of precision wireless thermometers	A. Javadpour; H. Memarzadeh-Tehran; F. Saghafi	2015	Title Review
Using Raspberry Pi to Create a Solution for Accessing Educative Questionnaires From Mobile Devices	R. A. RodrÃ-guez; P. Cammarano; D. A. Giulianelli; P. M. Vera; A. Trigueros; L. J. Albornoz	2018	Title Review

Mechanical Design of Dexterous Bionic Leg with Single-DOF Planar Linkage *	H. ZANG; D. LI; L. SHEN	2018	Title Review
The SOTM environment: Developments in satellite land, sea & air mobile terminals	M. Jarrold	2015	Title Review
Low-voltage ride-through enhancement with the Ï‰ and T controls of PMSG in a grid-integrated wind generation system	S. M. Tripathi; A. N. Tiwari; D. Singh	2019	Title Review
Network Function Virtualization: State-of-the-Art and Research Challenges	R. Mijumbi; J. Serrat; J. Gorricho; N. Bouten; F. De Turck; R. Boutaba	2016	Title Review
Systems Engineering Based Effective Approach for Executing Senior Projects for Engineering Students	H. El-Sherief	2019	Title Review
Digital geographic information based complex electromagnetic environment signal simulation and generation	Tai Xin; S. Liu; Fan Xiaoteng	2015	Title Review
Parallel Link-based Light-Weight Leg Design for Bipedal Robots	Y. Tazaki	2019	Title Review

A Feasible 5G Cloud-RAN Architecture with Network Slicing Functionality	C. Lee; M. Lee; J. Wu; W. Chang	2018	Title Review
A fine manipulation tweezer with embedded ultrasonic motor for assembly and gripping miniature parts	C. Li; B. Wu; S. Li; Q. Zhang	2016	Title Review
Reactive Microservices in Commodity Resources	D. Goel; A. Nayak	2019	Title Review
Ground-level observation of terrestrial gamma-ray bursts initiated by lightning	X. Li; R. Jiang; Y. Zheng; H. Zhou; B. Xing; Y. Li; W. Liu	2017	Title Review
High-Speed 2 Ã— 25 kV Traction System Model and Solver for Extensive Network Simulations	B. Mohamed; P. Arboleya; I. El-Sayed; C. GonzÃ¡lez-MorÃ¡n	2019	Title Review
Tunable Photonic Radio-Frequency Filter With a Record High Out-of-Band Rejection	P. Li; X. Zou; W. Pan; L. Yan; S. Pan	2017	Title Review
A cyber-physical architecture for industry 4.0-based power equipments detection system	M. Yu; M. Zhu; G. Chen; J. Li; Z. Zhou	2016	Title Review
Book of abstracts		2016	Title Review

MD2-NFV: The case for multi-domain distributed network functions virtualization	R. V. Rosa; M. A. S. Santos; C. E. Rothenberg	2015	Title Review
Total Ionizing Dose Effects on a Highly Integrated RF Transceiver for Small Satellite Radio Applications in Low Earth Orbit	J. Budroweit; M. Sznajder	2018	Title Review
MAVEN relay operations	N. Chamberlain; R. Gladden; P. Barela; L. Epp; K. Bruvold	2015	Title Review
In-Situ TID Testing and Characterization of a Highly Integrated RF Agile Transceiver for Multi-Band Radio Applications in a Radiation Environment	J. Budroweit; M. P. Jaksch	2019	Title Review
A comparison between several Software Defined Networking controllers	A. L. Stancu; S. Halunga; A. Vulpe; G. Suciu; O. Fratu; E. C. Popovici	2015	Title Review
Sales configuration creation for complex telecommunication solutions	T. GreguroviÄ‡; R. Penco	2018	Title Review
iCAST 2017 proceedings		2017	Title Review

2018 Index IEEE Robotics and Automation Letters Vol. 3		2018	Title Review
Table of Contents		2018	Title Review
Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems: An Industrial Experience Report	Li, Heng; Chen, Tse-Hsun (Peter); Hassan, Ahmed E.; Nasser, Mohamed; Flora, Parminder	2018	Title Review
Automated Grading of Collaborative Software Engineering Training with Cloud Distributing Scripts	Ma, Kun; Yang, Bo; Liu, Kun	2019	Title Review
FSE 2016: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering		2016	Title Review
ICFP 2016: Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming		2016	Title Review
Improve Student Performance Using Moderated Two-Stage Projects	Chen, Juan; Cao, Yingjun; Du, Linlin; Ouyang,	2019	Title Review

	Youwen; Shen, Li		
Multidisciplinary Groups Learning to Develop Mobile Applications from the Challenge Based Learning Methodology	da Costa, Andrew Diniz; de Lucena, Carlos Jos\'{e} Pereira; Coelho, Hendi Lemos; Carvalho, Gustavo Robichez; Fuks, Hugo; Venieris, Ricardo Almeida	2018	Title Review
On the Use of Metaprogramming and Domain Specific Languages: An Experience Report in the Logistics Domain	Costa, Pedro Henrique Teixeira; Canedo, Edna Dias; Bonif\'{a}cio, Rodrigo	2018	Title Review

Parameterized Diamond Tiling for Stencil Computations with Chapel Parallel Iterators	Bertolacci, Ian J.; Olschanowsky, Catherine; Harshbarger, Ben; Chamberlain, Bradford L.; Wonnacott, David G.; Strout, Michelle Mills	2015	Title Review
Research on Distributed Database Access Technology Based on .NET	Dinghua, He	2018	Title Review
SA 15: SIGGRAPH Asia 2015 Mobile Graphics and Interactive Applications		2015	Title Review
Source-Code Similarity Detection and Detection Tools Used in Academia: A Systematic Review	Novak, Matija; Joy, Mike; Kermek, Dragutin	2019	Title Review
Towards Versioning of Arbitrary RDF Data	Frommhold, Marvin; Piris, Rub\'{e}n Navarro; Arndt, Natanael; Tramp,	2016	Title Review

	Sebastian; Petersen, Niklas; Martin, Michael		
Verdict machinery: ON the need to automatically make snese of test results		2016	Title Review
A design of a small mobile robot with a hybrid locomotion mechanism of wheels and multi-rotors	K. Tanaka; D. Zhang; S. Inoue; R. Kasai; H. Yokoyama; K. Shindo; K. Matsuhiro; S. Marumoto; H. Ishii; A. Takanishi	2017	Abstract Review
A Survey on Systems Engineering Methodologies for Large Multi-Energy Cyber-Physical Systems	E. Azzouzi; A. Jardin; D. Bouskela; F. Mhenni; J. Choley	2019	Abstract Review
Back to Basics - Redefining Quality Measurement for Hybrid Software Development Organizations	S. Pradhan; V. Nanniyur	2019	Abstract Review

Container Orchestration Engines: A Thorough Functional and Performance Comparison	I. M. A. Jawarneh; P. Bellavista; F. Bosi; L. Foschini; G. Martuscelli; R. Montanari; A. Palopoli	2019	Abstract Review
Design and Performance Analysis of a Nonstandard EPICS Fast Controller	J. Jugo; M. Eguiraun; I. Badillo; I. Arredondo; D. Piso	2015	Abstract Review
Development of information and communications technology related products: Technical expertise, infrastructures and processes	Ã–. Aydin	2016	Abstract Review
Driving self-learning system based on the virtual reality	D. Sun; X. Liu	2017	Abstract Review
Enhancing Product and Service Capability Through Scaling Agility in a Global Software Vendor Environment	R. Lal; T. Clear	2018	Abstract Review

Green Propellant Infusion Mission (GPIM) space vehicle integration and test status	W. Deininger; S. Plaisted; A. Sexton; T. Smith; V. Moler; M. Goldman; G. Simmons; D. Cavender; R. Osborne; R. Wendland; J. Jonaitis; D. Smith; L. Wotruba; M. Riesco; C. McLean	2016	Abstract Review
How to Make Business Intelligence Agile: The Agile BI Actions Catalog	R. Krawatzeck; B. Dinter; D. A. P. Thi	2015	Abstract Review
Implementation of a software application for comprehensive monitoring of children and young patients under closed regimes in Argentina (SISP)	M. C. Abeledo; F. Bruschetti; D. Priano; R. Bevilacqua; G. Aguilera; P. Iriso; D. MartÃ-nez; E. Abete; A. Lacapmesure;	2015	Abstract Review

	N. M. Calcagno; G. Altobelli		
OSI Standards and the Top Fallacy of Distributed Computing	J. Y. Shi	2016	Abstract Review
Refinement and Resolution of Just-in-Time Requirements in Open Source Software: A Case Study	A. Q. Do; T. Bhowmik	2017	Abstract Review
Research on visual navigation algorithm of AGV used in the small agile warehouse	W. Chun-Fu; W. Xiao-Long; C. Qing-Xie; C. Xiao-Wei; L. Guo-Dong	2017	Abstract Review
SecFT-SDN: Securing the Flow-Table for Software-Defined Network	R. You; B. Tu; Z. Yuan; J. Cheng	2019	Abstract Review
Smart Audio Sensors in the Internet of Things Edge for Anomaly Detection	M. Antonini; M. Vecchio; F. Antonelli; P. Ducange; C. Perera	2018	Abstract Review

Smart City IoT: Smart Architectural Solution for Networking, Congestion and Heterogeneity	L. Pawar; R. Bajaj; J. Singh; V. Yadav	2019	Abstract Review
The Increasing Importance of Utilizing Non-intrusive Board Test Technologies for Printed Circuit Board Defect Coverage	M. R. Johnson	2018	Abstract Review
Wireless SDN architecture Testbed to support IP Multimedia Subsystem	A. Issa; N. Hakem; N. Kandil	2019	Abstract Review
A Global View on the Hard Skills and Testing Tools in Software Testing	Florea, Raluca; Stray, Viktoria	2019	Abstract Review
Have Your Data and Query It Too: From Key-Value Caching to Big Data Management	Borkar, Dipti; Mayuram, Ravi; Sangudi, Gerald; Carey, Michael	2016	Abstract Review
A Global View on the Hard Skills and Testing Tools in Software Testing	R. Florea; V. Stray	2019	Article Review
A parallel computing model for container terminal logistics	B. Li; W. Shen	2015	Article Review
Development of an information platform for observation of seismic events in Chile using RAD methodology	M. V. Tombolini Echeverria; S. G. Cornejo; F. A. Pontigo	2016	Article Review

LTE-based MCPTT Architecture for Next Generation Railway Dispatching Communication System	J. Huang; J. Ding; Z. Zhong; B. Sun; W. Wang; K. Li	2018	Article Review
On the RESTful Web Services for Managing Application Virtualization Environments	E. C. Yildiz; E. Unal; H. Tuzun; D. E. Aktas; M. S. Aktas	2019	Article Review
Performance evaluation and improvement in cloud computing environment	O. Khedher; M. Jarraya	2015	Article Review
Set-based Design in Agile Development: Developing a Banana Sorting Module - A Practical Approach	D. Saad; S. RÃ¶tzer; M. Zimmermann	2019	Article Review
Systematic mapping study on MBT: tools and models	M. Bernardino; E. M. Rodrigues; A. F. Zorzo; L. Marchezan	2017	Article Review
The effect of software programmers' personality on programming performance	X. Li; P. Shih; E. David	2018	Article Review
Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges	Shahin, Mojtaba; Babar, Muhammad Ali;	2017	Article Review

	Zahedi, Mansooreh; Zhu, Liming		
In-Memory Integration of Existing Software Components for Parallel Adaptive Unstructured Mesh Workflows	Smith, Cameron W.; Granzow, Brian; Ibanez, Dan; Sahni, Onkar; Jansen, Kenneth E.; Shephard, Mark S.	2016	Article Review
Key Factors in Scaling up Agile Team in Matrix Organization	Gupta, Rajeev Kumar; Jain, Shivani; Singh, Bharat; Jha, Sanjay Kumar	2019	Article Review
Managing Quality Assurance Challenges of DevOps through Analytics	Ibrahim, Mahmoud Mohammad Ahmad; Syed-Mohamad, Sharifah Mashita; Husin, Mohd Heikal	2019	Article Review

APPENDIX D:
RESEARCH QUESTION MAPPING

Below is a mapping of all research questions within the papers analyzed (if there were any) and their answers. In all cases, the questions in the question column are direct quotes from the paper itself and should be thought of as such. In some cases, the answers to the research questions are paraphrases from the papers in which they are found but are often direct quotes as well. The author does not claim that any text in the following table is original content but drawn from the adjacent paper designated in the Id column.

Id	**RQ Number**	**Question**	**Answer**
S6	1	How often does test automation appear in open-source Android projects?	about 47% of the time
S6	2	Which testing frameworks are adopted in open-source Android projects?	Android.Test, EasyMock, Fest, Hamcrest, JUnit, Roboelectric, Robotium, Espresso
S6	3	What is the relation between the elements of production code and the elements of test code?	About 8.3% LoC are dedicated to tests
S6	4	Do the automated tests of open source Android projects cover specific challenges of mobile devices, namely, connectivity, rich GUIs, limited	Connectivity: about 36% of the time Rich GUIs: about 30% of the time Limited Resources: 0%

		resources, sensors, and multiple configurations?	(no performance testing found) Sensors: About 19% of the time Multiple Configurations: about 48% coverage
S8	1	How expressive is the BDLT language in regards to load test concerns of industrial use cases?	Can express all use cases named by industrial partners, but had to make use of extensions mechanisms for one custom event
S8	2	How would BDLT be used in industrial contexts?	By DevOps teams to define load tests
S8	3	What are the benefits and limitations of using BDLT in comparison to defining load test scripts?	Benefits: natural language Limitations: need for extensions for custom events, non-trivial subset of config possibilities
S16	1	Which type of tests a crowd tester is used to run	Functional, usability, and performance testing are most required Stress and load are least applied

S16	2	What are the challenges and limitations faced by crowd testing?	Timing and time pressure
S16	3	How laboratory testing and crowd testing can complement each other?	* Better communication * Showing more underlying info on SUT * Competition/pressure should be addressed
S22	1	How do developers navigate and what information and representation is supportive for locating a performance bug?	Real-time information regarding method calls and resource consumption is important to finding performance bugs, visually displaying this information is helpful, and developers will either toggle or path-follow through such information
S22	1.1	How was information from the profiling tool or other parts of the IDE used to locate the performance bug?	Using dynamic instances of method calls as links to runtime information
S22	1.2	Is the in-situ visualization of the profiling data beneficial compared to a traditional list representation?	Yes

S22	1.3	What navigation strategies do developers pursue to locate a specific performance bug?	Toggling - Switching back and forth between test classes and other important classes Path Following - Following dynamic calls through the visualization
S22	2	How do developers try to understand and explain the causes of performance bugs?	Most teams formulate a hypothesis early, discussed architecture and algorithms around the code, and seemed to find sketches useful
S22	2.1	How do developers communicate with each other when locating a performance bug?	Clear communication strategies were reportedly not detectable in this study
S22	2.2	Could sketches help understand and communicate a performance bug?	Sketching has obvious advantages in pair programming scenarios but its usefulness is unclear in a single-developer scenario

S23	A	To what extent is the system able to filter invalid messages?	This is dependent on the volume and frequency of the invalid messages
S23	B	Is delaying messages an effective strategy to discover faults?	Yes, helps find timeout-handling that has been configured incorrectly. This was not found in manual testing
S24	RQ	How can visualization support developers in the analysis of the impact of source code changes to the performance of a system?	(note: authors say this paper is an "initial step toward answering" this question) Visualization needs to be large enough to model a whole system, but small enough to fit on a screen Properties of text on visualization impacts usability Live visualization provides feedback on how current changes affect the system

S29	1	What are the most used practices in this context? (context of RE and ST alignment in agile)	Nonspecifically: weekly meetings with project stakeholders, conceptual Use Case models, description and execution of test cases, use of FitNess tables
S29	2	What techniques, strategies, and tools have been adopted?	Model V, a REST taxonomy, conceptual models, ATDD
S29	3	What are the main challenges encountered in the association of RE and ST?	for req engineering: elicitation and verification, changing management, maintaining documentation, un-useful Fit tables, maintaining artifacts. For software testing: mostly related to req engineering problems and reportedly solved via automation
S29	4	What are the open problems identified?	Open problems mostly related to test activity

			automation, validating use case effectiveness, maintain req documentation, consolidating req artifacts
S29	5	What requirements and software testing artifacts are generated?	Class, packet, state, activity, and use case diagrams. Business models, traceability matrices, user stories, and Fit tables
S32	First	what group is most likely to find either an in-house or field defect based on defect severity and test type?	System test team - most likely to find normal or major defect, or defect of any severity. Function test - most likely to find critical defects System test team - most likely to find system, functional, and combined defect type. Performance team - most likely to find

			performance defect Security team - most likely to find security defect Customer is most likely to find any defect of any severity
S32	Second	what is the field defect discovery rate during the first fourteen days of a release?	Peaks at 6% on day 5, then 5% on day 7
S2	1	What are the main objectives for cloud testing?	Performance, then functional, security, elasticity, and reliability
S2	2	How are cloud resources exploited for software testing?	Combinatorial testing is common, some algorithms are used to evaluate cloud performance under given configurations
S2	3	What are the test methods, techniques, and tools mainly used in cloud testing?	Test case generation, parallelization, MBT, combinatorial techniques. Many frameworks for testing in cloud exist, offering

			scaling features, mobile testing features.
S2	4	How are testing results evaluated in cloud testing?	Comparison of quality attributes for SUTs in different conditions are evaluated. Results of many tests shown via frameworks with web-based visualizations. Monitoring of test executions. Performance is the thing most often tested in the cloud
S2	5	What are the research issues and future research directions of cloud testing?	Frameworks for parallel test execution, effective/efficient resource allocation, elastic environments, speeding up tests, self-service testing, real-world emulation
S2	6	Which are the main application domains for software testing in the cloud?	Web and mobile applications, cloud infrastructural applications, SOAs

S3	1	What are the done criteria used in agile software development projects?	62 criteria identified, may be broadly grouped as criteria related activity, metrics, targets, standards, and checklists
S3	2	What are the characteristics of the application domain of the papers that report the done criteria identified in RQ1?	Most methods used are Scrum, product domain most often not presented, application domain is most often industry. Teams vary widely in both size and distribution
S3	3	What types of studies are performed in the papers the report the done criteria identified in RQ1?	Most often solution proposals using a method or means of development. Empirical types are usually Case study when there is one

APPENDIX E:

GLOSSARY

This section will contain some definitions of commonly used acronyms throughout the paper.

Acronym	Spelled Out	Definition
NFR	Non Functional Requirement	Describes expected criteria of system operation rather than behavior
LPT	Lulu Performance Test	The performance testing framework developed for this study
SUT	System Under Test	Refers to the software system being tested
CI	Continuous Integration	Software engineering practice in which developers check-in code regularly
CD	Continuous Delivery	Software engineering practice in which a system is always ready to be deployed
CDE	Continuous Deployment	Software engineering practice in which a system is deployed as soon as changes are checked in and tests are passed (not used in this study but

		recorded to highlight that CDE is not CD)
CI/CD	Continuous Integration/Continuous Delivery	
ASD	Agile Software Development	Collection of software development methodologies in which software is developed incrementally
SLR	Systematic Literature Review	A kind of research study in which a topic is studied in a reproducible way
RQ	Research Question	Question that a research item endeavors to answer
SLR-RQ	Systematic Literature Review Research Question	
MBT	Model Based Testing	Testing practice which uses models rather than test steps
DSL	Domain Specific Language	Computer language used for a specific and likely narrow domain

GPL	General Purpose Language	Computer programming language such as Python, Java, Ruby, etc. (not used in this study but recorded to highlight difference between GPL and DSL)
YAML	YAML A'int Markup Language	Data serialization language. Used specifically in this study to configure TravisCI build scripts
POS	Point of Sale	A software system for managing sales and inventory
AWS	Amazon Web Services	Suite of cloud services provided by Amazon
ORM	Object Relational Mapper	Software that maps database tables and columns to source code classes and attributes
MVC	Model-View-Controller	Software architecture pattern popular in many web development frameworks
JSON	JavaScript Object Notation	Data serialization language originally designed to describe JavaScript classes, now has many uses
JAR	Java Archive	Executable Java package