MORGAN & CLAYPOOL PUBLISHERS

Hard Problems in Software Testing

Solutions Using Testing as a Service (TaaS)

Scott Tilley
Brianna Floss

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

Hard Problems in Software Testing

Solutions Using Testing as a Service (TaaS)

Synthesis Lectures on Software Engineering

The Synthesis Lectures on Software Engineering publishes 75-150 page publications on all aspects of software design, engineering, and process management.

Hard Problems in Software Testing: Solutions Using Testing as a Service (TaaS)
Scott Tilley and Brianna Floss
August 2014

T Model-Driven Software Engineering in Practice itle
Marco Brambilla, Jordi Cabot, Manuel Wimmer
September 2012

Hard Problems in Software Testing: Solutions Using Testing as a Service (TaaS)
Scott Tilley and Brianna Floss
www.morganclaypool.com

ISBN: 9781627055239 print
ISBN: 9781627055246 ebook

DOI 10.2200/S00587ED1V01Y201407SWE002

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES TITLE ON SOFTWARE ENGINEERING # 2

Series ISSN 2328-3319 Print 2328-3327 Electronic

Hard Problems in Software Testing

Solutions Using Testing as a Service (TaaS)

Scott Tilley
Florida Institute of Technology

Brianna Floss
Lockheed Martin

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING #2

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book summarizes the current hard problems in software testing as voiced by leading practitioners in the field. The problems were identified through a series of workshops, interviews, and surveys. Some of the problems are timeless, such as education and training, while others such as system security have recently emerged as increasingly important.

The book also provides an overview of the current state of Testing as a Service (TaaS) based on an exploration of existing commercial offerings and a survey of academic research. TaaS is a relatively new development that offers software testers the elastic computing capabilities and generous storage capacity of the cloud on an as-needed basis. Some of the potential benefits of TaaS include automated provisioning of test execution environments and support for rapid feedback in agile development via continuous regression testing.

The book includes a case study of a representative web application and three commercial TaaS tools to determine which hard problems in software testing are amenable to a TaaS solution. The findings suggest there remains a significant gap that must be addressed before TaaS can be fully embraced by the industry, particularly in the areas of tester education and training and a need for tools supporting more types of testing. The book includes a roadmap for enhancing TaaS to help bridge the gap between potential benefits and actual results.

KEYWORDS

software testing, hard problems, testing as a service (TaaS)

Contents

Figures

Tables

Foreword

Some people think that "hard problems in software testing" is a non sequitur. However, those of us that have done software testing know just how true it is to say, "the testing of software is a very challenging endeavor."

I commend Scott Tilley and Brianna Floss for taking on the difficult task of researching, analyzing, and reporting on the topics in this book. This kind of work is much needed in our industry because there are indeed hard problems in software testing—problems affecting all of us. The specific testing problems may change, for example, due to the introduction of new approaches such as service orientation or new technologies such as cloud computing, but the essential nature of testing is timeless.

I was one of the practitioners that attended an event on the campus of the Florida Institute of Technology, in the early days of this research, to discuss hard problems in software testing. It was obvious to me that many in this industry are dealing with the same issues and this book has captured those issues very well. I encouraged other testing professionals at Lockheed Martin to participate in their survey so that our views would be captured as part of the study.

Scott and Brianna's research into Testing as a Service (TaaS) is also a much-needed endeavor. Their research has resulted in findings that should be very useful to any group thinking about using TaaS. Many organizations around the world, from both government and industry, are looking to change the paradigm for software testing. TaaS has the potential of achieving an improved approach to testing software while also increasing the quality of the testing results.

I know that upon reading this book you will find the research presented not only enlightening, but useful as well. Like me, I am sure you will also want to encourage Scott and Brianna to continue the "future work" they mention.

Thomas L. Wissink
Senior Fellow
Director of Integration, Test & Evaluation (IT&E)
Lockheed Martin Corporate Engineering

Preface

Software testing is a crucial phase of the software engineering lifecycle, responsible for assuring that the system under test meets quality standards, requirements, and consumer needs. Unfortunately, as software systems grow in size and complexity, testing becomes commensurately challenging and quality becomes significantly more difficult to ensure.

Some of the hard problems in software testing are timeless, while others are brought on by the introduction of new technologies and methodologies. For example, cloud computing offers elastic resources that can be leveraged for software testing, but cloud-based applications introduce new intricacies into the testing process. Similarly, agile development places unique strains on testing activities—particularly in terms of compressed schedules.

The *Hard Problems in Software Testing* (HPST) project addressed this issue by asking real testers a relatively simple question: "Where are your pain points?" We hoped that their candid answers would help us (researchers) help them (practitioners) by directing our efforts on issues that truly matter to them. It is our sincere belief that working together, we can make headway toward solving some of the grand challenges in software testing.

WHAT IS UNIQUE ABOUT THIS BOOK?

We began the HPST project in Fall 2011 with an interactive session held as part of the *3rd International Workshop in Software Testing in the Cloud* at the IBM CASCON conference in Toronto, Canada. We then hosted an invitational symposium at the Florida Institute of Technology and organized a workshop at the Software Testing Analysis & Review (StarEast) conference in Orlando in Spring 2012.

In addition to the HPST series of events, we continually gathered data using an online survey and selected personal interviews. The project's goal was to identify hard problems in software testing as voiced by leading practitioners in the field. These problems had to be sufficiently challenging to warrant focused attention for the next five years. They also had to be sufficiently important to the community such that if measurable progress were made in solving these problems, we would advance the state of the practice in software testing to everyone's benefit.

Software testing is both practically relevant and academically interesting. It is a somewhat unique software engineering activity area that is both very applied and well-grounded in sound computing theory. Unfortunately, there has traditionally been a schism in testing between new approaches espoused by researchers and the timeless problems that practitioners struggle with on a

daily basis. Such disconnects can hinder communication between the two camps and thereby limit the adoption of promising new testing tools and techniques.

WHO SHOULD READ THIS BOOK?

This book represents the culmination of a three-year effort. In addition to summarizing our findings of the current hard problems in software testing, this book also examines where Testing as a Service (TaaS) may offer a viable solution to these important problems. TaaS is a relatively new model of software testing that relies on cloud computing and service orientation to provide testing capabilities unavailable to many organizations. Researchers may benefit from an introduction to TaaS and glean insights as to where this new approach to testing could aid in some of their own projects.

Software testing practitioners (including test managers) may find solace that many of the hard problems they struggle with are shared with others in multiple disciplines. The evaluation of representative TaaS tools could aid in their own decision making process as they consider alternate solutions to their testing challenges.

Students are always looking for thesis and dissertation topics. This book is full of possible research themes, as evidenced by the hard problem summary. The gap analysis between perceived testing need and current TaaS capabilities also provides a number of promising avenues of future work. There is certainly no lack of topics to choose from.

OUTLINE OF THE BOOK

The book is structured as three main parts: a summary of the hard problems in software testing, an examination of testing as a service, and case study with gap analysis. Chapter 2 focuses on the Hard Problems in Software Testing project, explaining its background, the process, the results, and some conclusions. Chapter 3 details the field of Testing as a Service and explores the academic and commercial state of the industry. Chapter 4 offers a case study, evaluating a selection of TaaS tools against criteria developed from the software testing challenges, and analyzes the gap between the current state of the industry and the needs of testing professionals.

Scott Tilley and Brianna Floss
Melbourne, FL
July 2014

Acknowledgments

We would like to thank all the participants in the HPST events, interviews, and surveys. Without their willingness to share their experiences with us, our findings would be limited to our own viewpoints rather than a representative cross-section of the industry.

Special thanks go to Tauhida Parveen for her help in organizing some of the HPST activities.

Thank you to the book's reviewers for the invaluable comments. Your suggestions helped improve the text. Any remaining errors or omissions are ours.

We are grateful to the Florida Institute of Technology for supporting this research.

Finally, thank you to Morgan & Claypool for helping us publicize the results of our work.

Dedication

To Miel
— Scott Tilley

To my family
— Brianna Floss

List of Abbreviations

API	Application Programming Interface
APM	Application Performance Monitoring
ART	Average Response Time
BP	Basic Path
BRO	Branch and Relational Operator
CKMS	Collaborative Knowledge Management System
CSS	Cascading Style Sheet
CTOMS	Cloud Testing of Mobile Systems
EC2	Amazon Elastic Compute Cloud
FaaS	Failure as a Service
GAE	Google App Engine
GEO	Generalized Extremal Optimization
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IaaS	Infrastructure as a Service
INCOSE	International Council on Systems Engineering
JPQL	Java Persistence Query Language
JSP	JavaServer Pages
NIST	National Institute of Standards and Technology
OVA	Open Virtualization Archive
PaaS	Platform as a Service
REST	Representational State Transfer
RTS	Regression Test Selection
SaaS	Software as a Service
SDK	Software Development Kit
SEI	Software Engineering Institute
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
STTIC	Software Testing in the Cloud
TaaS	Testing as a Service

WSDL	Web Services Description Language
XML-RPC	Extensible Markup Language – Remote Procedure Call

CHAPTER 1

Introduction

Software testing is "an empirical technical investigation conducted to provide stakeholders with information about the quality of a product or service under test" [1]. Testing is a critical phase of the software testing lifecycle, paramount to guaranteeing that the system under test meets quality standards, requirements, and consumer needs. While the field has existed for over half a century, software testing is not without challenges, difficulties, and flaws. Some are brought on by new and evolving technologies and methodologies while others are timeless, existing since the field's inception.

1.1 SOFTWARE TESTING CHALLENGES

As software systems grow in both size and complexity, quality (defined as having "value to some person" [1]) becomes significantly more difficult to ensure. These increasingly complex systems make existing testing issues more prominent and cause new issues to arise. Testing has not been ignored by the research community, but traditional research topics focus on only a small portion of these challenges, many of which may or may not be the most crucial issues for the industry as a whole.

Donald Firesmith of Carnegie Mellon University's Software Engineering Institute (SEI) detailed a series of recurring problems that he identified during his three decades of software development and independent technical assessments of development projects. Originally published in the *SEI Blog* [5, 6] and related presentations [7] and culminating in a recent book [9], he analyzed his most commonly occurring software testing problems as derived from both his own experienced and feedback from 29 members of various LinkedIn groups and the International Council on Systems Engineering (INCOSE), incorporating their findings and weighing them against his own.

Firesmith separated these findings into two primary categories: general testing problems and test type-specific problems. General testing problems were further divided into eight sub-categories: test planning and scheduling problems, stakeholder involvement and commitment problems, management-related testing problems, test organizational and professionalism problems, test process problems, test tools and environments problems, test communication problems, and requirements-related testing problems.

Although the identified problems are common to some or many projects, Firesmith's work was not prioritized by frequency or severity. Furthermore, several of his recommendations detailed in his presentation [8] oversimplify the problem and, in some cases, provide solutions that would not be achievable in many environments. For example, if the test team has inadequate experience,

his suggestion is to hire professional testers and provide more training—both of which can be difficult or impossible to achieve with budgetary and time constraints.

Our approach to identifying and quantifying the major software testing challenges currently facing the industry is somewhat different. The software testing challenges were obtained from industry professionals through the Hard Problems in Software Testing (HPST) series of events [3], personal interviews, and an online survey [4]. The survey queried respondents about issues they have faced in their work and on which areas they believed researchers should focus their attention. Results were gathered and analyzed quantitatively for recurrences. The responses were analyzed both individually and in categories of similar statements. More details on the HPST project are provided in Chapter 2.

1.2 SERVICE-ORIENTED SOLUTIONS

Many tool vendors are promoting a service-oriented solution, commonly known as Testing as a Service (TaaS), to address software testing challenges. TaaS is a model of software testing that leverages the vast and elastic resources of the cloud to offer easily accessible services that handle a variety of testing activities for consumers on a pay-per-test basis [2]. The move toward TaaS is driven by several recurring issues in software testing, including increasing costs and effort, limited time, and difficulty handling a large variety of testing.

In theory, TaaS could resolve some of the major issues plaguing the industry. In practice, it is unclear where TaaS is applicable and how effective it can be for various software testing problems. Leah Riungu-Kalliosaari, a researcher and doctoral student at Finland's Lappeenranta University of Technology (now at the University of Helsinki), studied software testing in the cloud [8]. Specifically, she focused on many of the socio-economic factors that influenced the adoption and use of cloud-based testing in a variety of organizational contexts in order to provide a better framework of cloud-based testing and propose strategies that could assist organizations in adopting these cloud-based testing technologies. Beginning in 2010, she and her collaborators, Ossi Taipale and Kari Smolander, conducted a series of interviews with members of the industry focused on Testing as a Service.

These interviews were performed with individuals working in consumer and provider roles in TaaS at various companies and served to explore conditions affecting the adoption of TaaS [11], elicit research issues [12], and discern the effects of cloud-based testing [13]. The interviews consisted of semi-structured questions. All responses were recorded and analyzed using ATLAS.ti, a qualitative data analysis tool. Ultimately, Riungu-Kalliosaari published a series of papers providing benefits, challenges, and needs of TaaS, as well as a roadmap for organizations considering the feasibility of TaaS migration. The results of this research are included in Chapter 3.

While Riungu-Kalliosaari's work is important to those adopting or improving TaaS technologies, her work did not directly answer the question proposed by this book: How software testing

challenges can be resolved by utilizing TaaS. The work on TaaS adoption aligned with several of the challenges identified by the HPST survey, but this was, in many cases, coincidental.

Our approach to determining the applicability and efficacy of TaaS to today's testing challenges is built upon a literature study of the academic research and commercial offerings of TaaS tools and techniques. This survey provides a baseline of understanding of current TaaS solutions. Details of the survey are provided in Chapter 3.

1.3 GAP ANALYSIS BETWEEN HPST AND TaaS

To truly ascertain if TaaS is an appropriate solution to a specific testing problem, more information is needed. We obtain this information by performing a gap analysis between the requirements implicit in the hard problems in software testing and available TaaS solutions. The gap analysis is primarily qualitative in nature, but it does offer insight into the possible solution space provided by TaaS.

Chapter 4 presents a case study of a web application and discusses its testing needs with three representative TaaS solutions. The case study embodies several hard problems in software testing. General observations are cautiously made from the case study as to the broader applicability of TaaS to HPST.

The gap analysis also identifies possible improvements to TaaS. These unmet requirements are suitable starting points for future research. For the practitioner, these unmet requirements are warnings that very few hard problems have single-point solutions. We expect that it will always be axiomatic in software testing that a federation of tools and techniques are needed to fully address hard problems.

CHAPTER 2

Hard Problems in Software Testing

Software testing is a critical phase of the software lifecycle, responsible for assuring the quality of a system under test; however, testing suffers from a variety of challenges—both new and timeless. In order to evaluate the ability of TaaS to overcome these challenges, the needs of the industry must first be identified.

In academia, a significant portion of testing-related research seems esoteric in nature. While the topics themselves are fascinating, the niche focus provides value to only a small portion of the field. The option arose to broaden the view and explore problems as they exist on a more universal scale. Many software testing challenges are well-known, especially those considered timeless; however, they need to be quantified to better determine which solutions could provide the most benefit to the industry.

The Hard Problems in Software Testing (HPST) survey was conducted to elicit the current issues plaguing the field in a quantifiable manner. Many software testing problems are challenging enough to demand sufficient research in the near future and important enough to the community that their resolution could advance the industry as a whole.

2.1 THE HPST SURVEY

The objective of the HPST survey was to obtain and quantify challenges that exist in the industry today, as encountered by testing professionals. The goal was to offer these findings as motivation for future research focused on the needs of the industry as a whole. The solutions themselves are far more complex than the issue at hand and were not considered as part of the initial survey effort.

The HPST survey was written in late 2011. The original survey is available in Appendix A. The following questions were asked:

- Tell us about yourself (name, affiliation, job title, email, and phone number).
- Describe five software problems you have struggled with in your job (these could be related to process, tools, education & training, and so on).
- Are you working in any of the following areas as a tester?
  - Service-Oriented Architecture (SOA)
  - Security/Information Assurance
  - Cloud Computing

- Mobile/Apps

- What topics do you think the research community should focus on over the next five years to help you do your job better?
- Are you interested in collaborating with us? If so, on what problem area? In what capacity?
- Any additional comments.

The survey went live on SurveyMonkey [3] in November 2011 and was promoted on software testing forums. For the purpose of this research, answers were collected until November 2012. During that same time period, the survey was offered at seminars on Hard Problems in Software Testing at IBM CASCON 2011, STAR*EAST* 2012, the 2012 Software Testing in the Cloud (STITC) Workshop, and at Florida Institute of Technology.

A total of sixty-four individuals participated in the HPST survey. Of these sixty-four, forty were industry professionals surveyed at conferences and on SurveyMonkey. These respondents represented over twenty companies including Lockheed Martin, Intel, Yahoo, and SAP. The experience of industry professionals is shown in Table 2.1. The four specific areas (SOA, security, cloud, mobile) were addressed in the survey because it was felt that they represented some of the key topics that testers are (or soon will be) concerned with.

Table 2.1: Areas of testing experience

Area	Frequency
Service-Oriented Architecture (SOA)	16
Security / Information Assurance	13
Cloud Computing	8
Mobile / Apps	10

Considered separately, the other twenty-four were master and doctoral candidates in computer science, software engineering, and other related fields at Florida Institute of Technology. These students were surveyed during a seminar on Hard Problems in Software Testing in 2012. A small percentage had sufficient industry experience to answer questions in the same vein as industry professionals. The remaining responses demanded that the student surveys be considered independently of the industry professionals' surveys.

2.2 SURVEY RESULTS

For the purpose of this research, the second and fourth questions were examined in detail. Using a qualitative method called grounded theory [14], the results were explored systematically. The individual responses were repeatedly examined until they were all coded, allowing them to be placed

within categories. The responses are shown in the following sections, organized by these categories and ordered by frequency. The results are discussed in Section 2.3.

2.2.1 SOFTWARE TESTING PROBLEMS

Question: "Describe five software problems you have struggled with in your job."

The results of the HPST survey's second question concerning testing problems with which individuals have struggled can be divided into several focus areas: testing, test data, personnel, metrics, estimation, process, platforms/environment, security, and tools. Testing contains the following subcategories: manual, automated, regression, unit, and other. Figure 2.1 shows the breakdown of responses per category.

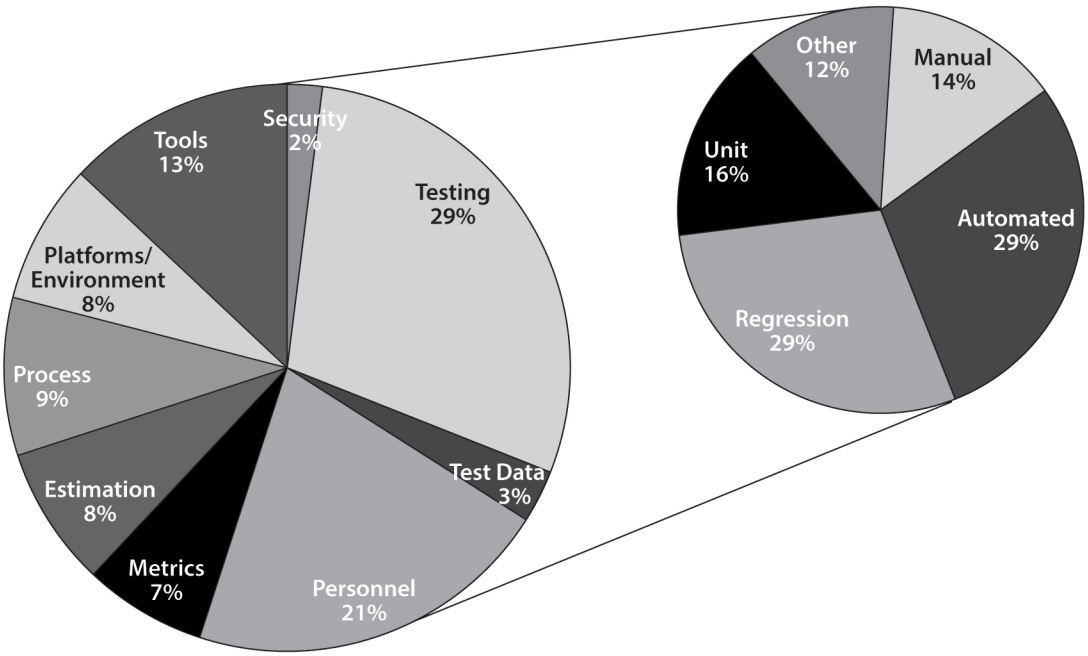


Figure 2.1: Breakdown of responses per focus area.

The individual responses to this question are listed by category in the Tables 2.2–2.10. For each response, the frequency of that response—the number of individuals that responded with the same statement—is included on the right.

Table 2.2: Testing problems and frequency

Testing Problems	Frequency
Manual Testing	
Need to create a more efficient/effective testing process	2
Manual tests are not repeatable	2
Tests take too long to setup and run	2
Testing too time consuming	1
Automated Testing	
Difficulty setting/starting up automated testing	4
Automated testing underdeveloped or underused	2
Generated tests never fully utilized	2
Concerns of affordability (time and cost)	2
Need to reduce feedback time from automated tests	2
Need to run a very large number of test cases	1
Determining what should be automated	1
Inability to automate for complex systems	1
Regression Testing	
Cost to write and maintain tests	5
Difficulty maintaining tests	3
Misconceptions or misuse of regression testing	3
Incomplete regression test suites	2
Inability to store large amounts of test data	1
No existing regression tests	1
Unit Testing	
Unit testing not performed or insufficient	4
Need to motivate developers to engage in unit testing	3
Unit testing difficult for web applications	1
Other	
Need for performance testing	2
Need for security testing	2
Need for uniform certification testing	1
Difficulties starting up Testing as a Service	1

Table 2.3: Test data problems and frequency

Test Data Problems	Frequency
Difficulty managing large sets of test data	3
Need to maintain data privacy	1
Difficulty generating realistic and unique sets of data	1

Table 2.4: Personnel problems and frequency

Personnel Problems	Frequency
Lack of education/experience in testing methods · Lack of education in creating tests	12
Need to improve communication and coordination between test team and other groups	4
Developers acting as testers	2
Lack of knowledge on system under test	2
Complex systems require more training	2
Difficulty keeping up with current technologies	2
Turnover rates	2
Need for better methods of knowledge sharing between experienced and novice employees	2
Test team feels underappreciated	2
Getting people to understand the importance of testing	2
Roles of test team need to be better defined	1
Lack of testers	1
Difficulty recruiting talent	1
Lack of programming skills	1
Need for tester certification	1

Table 2.5: Metrics problems and frequency

Metrics Problems	Frequency
Determining test coverage	3
Finding powerful and valuable metrics	3
Need to write better bug reports	3
Need to quantify defects	2
Determining how much coverage is required	1
Metrics nonexistent or not maintained	1

Table 2.6: Estimation problems and frequency

Estimation Problems	Frequency
Creating realistic schedules	5
Need help with estimation	3
Generating realistic budgets	2
Budgeting for tools, training, and setup	2
Inability to determine cost-benefit for cloud migration	2

Table 2.7: Process problems and frequency

Process Problems	Frequency
Too much/all of testing at end of cycle	5
Need process for testing in agile environments	4
Nonexistent, poor, or lacking procedures or test plans	2
Better management of testing needed	2
Lots of exit criteria but limited or no entry criteria	2
Process not documented	1
Need for continuous integration	1

Table 2.8: Platforms/environment problems and frequency

Platforms/Environment Problems	Frequency
Environment scenarios unrealistic	3
Lack of adequate simulators	3
Keeping up with technology/evolution	2
Maintaining up-to-date platforms	2
Shortage of testing hardware	1
Resources limited	1
Variations in platforms produce inconsistent results	1
Environments need better management	1

Table 2.9: Security problems and frequency

Security Problems	Frequency
Maintaining data privacy and security, especially when outsourcing testing	2
Security concerns when testing	1

Table 2.10: Tools problems and frequency

Tools Problems	Frequency
Need for tools or need for better tools	8
Tools need to be:	
· Adaptable to new technology	3
· Better documented or packaged with training material	2
· Easier to learn/simpler	2
· Properly tested	1
· Sturdier	1
· Able to meet rigorous security demands	1
Inconsistency in tools (different tools needed for different languages)	1
Lack of tools for existing baseline	1
Need help evaluating tools	1
Vendors need to be knowledgeable and trustworthy to help guide the user community	1

Students without industry experience offered a different set of problems, highly influenced by software testing classes and their software development coursework. Some of the stated challenges were universal, aligning closely with the industry responses. Examples include time constraints, considering all testing criteria, and compatibility between applications, tools, and versions. Others were far more focused on development, debugging, and self-testing of the students' own code, such as null pointer exceptions, array out of bounds exceptions, memory errors, and validity of inputs. While the similarities and differences in software testing problems were interesting, further discussion in this research is limited to responses from industry professionals.

2.2.2 TOPICS FOR THE RESEARCH COMMUNITY

Question: "What topics do you think the research community should focus on over the next five years to help you do your job better?"

The HPST survey also questioned individuals about what topics they believed that the research community should focus on in the near future. The results varied, but generally followed a similar pattern as the previously discussed question on the software testing problems. Recommended research topics are listed in Tables 2.11–2.16 and are categorized as follows: testing, personnel, estimation, tools, standards, and miscellaneous areas of interest. Student responses were sparse and were not considered in the following tables.

Table 2.11: Testing research topics

Testing Topics	Frequency
How to improve efficiency and effectiveness of the testing process	4
Agile testing: How to adequately test a system without testers	3
How to improve regression testing	2
Automated/Regression testing: How to reduce the cost of testing	2
How much should be automated and when to automate	1
What is the minimal set of devices that should be tested	1
Guidelines for uniform certification testing	1

Table 2.12: Personnel research topics

Personnel Topics	Frequency
How to train/certify testers	4
How to enforce knowledge sharing amongst employees	1
Guidelines for testing education	1
How to improve communication between the test and other teams	1

Table 2.13: Estimation research topics

Estimation Topics	Frequency
How to estimate effort	1
How to estimate risk	1
How to estimate quality	1

Table 2.14: Tools research topics

Tools Topics	Frequency
Better software testing tools	2
What tools are needed and what tools already exist for specific testing purposes	1

Table 2.15: Standards research topics

Standards Topics	Frequency
Creating standards for cloud computing	1
Creating standards for mobile testing	1

Table 2.16: Miscellaneous research topics

Miscellaneous Topics	Frequency
Creating a common infrastructure to prevent reinventing the wheel	1
Knowledge sharing of frequently identified bugs in embedded testing	1
Improving overall system security	1
How to bring qualitative methods to the masses	1
Improving data handling	1
Providing better methods of simulating time	1
How to improve Service Level Agreements	1

2.3 DISCUSSION OF RESULTS

The HPST survey offered a qualitative approach to identifying some of the top challenges the industry faces today. From the gathered responses, the top items were examined more closely and relationships between them analyzed. Additionally, the most desirable research areas become obvious and oftentimes tie closely to the aforementioned challenges.

2.3.1 SOFTWARE TESTING PROBLEMS

While a large number of software testing problems were identified over the course of the survey, the top five were easily established. Worthy of note, the vast majority of these areas could be considered timeless challenges—those existing since software testing's inception. The following represent the most commonly observed issues in software testing today:

1. A desire for education and training

2. A need for testing tools
3. Lacking, insufficient, or nonexistent testing
4. Generating realistic schedules
5. A deficiency in communication

The single most commonly discussed software testing problem was tester education and training. Testing is a broad and constantly evolving field. Truly comprehending every aspect of it is a lofty goal—one that may be unachievable for the majority of working professionals. Nevertheless, testers should endeavor to become a "jack of all trades" within their field. Training, certification, and education can all help, but common standards for these three areas do not exist. Furthermore, determining how much training is necessary and on which topics testers should focus is no easy task, especially on a universal scale.

Beyond training, respondents frequently cited a need for tools or a need for better tools. A wide variety of software testing tools exist but many of them lack flexibility, limiting the tool's usage to a small portion of projects, or are too complicated to comprehend and apply within the already restricted project schedule. Even if tools are found that can cover certain aspects of testing, the team must spend time and effort learning and using the various tools in a manner that meets their needs. This process can be quite expensive, especially after factoring in the cost of tools.

Issues with testing itself made up the third most common testing problem. Specifically, respondents described lacking, insufficient, or nonexistent testing, particularly in the areas of automated, regression, and unit testing. A major factor contributing to this challenge is the aforementioned need for education and training—having the knowledge required to implement the various desired tests within schedule and budget. Another factor is having enough time allotted in the schedule during which tests can be written, updated, and performed.

Another timeless software testing problem is generating realistic schedules that prevent testing from falling at the very end of the project lifecycle. In traditional models of the software lifecycle, the testing phase falls last, dependent on the completion of the previous phases. With less flexible deployment deadlines, testing is further reduced by delays in the requirements, design, and development phases, leaving little time for assuring quality. Creating, and more importantly adhering to, a realistic schedule can provide the test team with much-needed time. Even so, testing still demands more time and attention than it typically receives. The best option is to perform testing in a more agile manner—as an integral part of development.

Finally, respondents felt that communication, both within the team and with other groups, was a major issue within software testing. The internal communication was primarily concerned with knowledge sharing between novice and experienced team members—another aspect of the lacking training and education. External communication included exchanges between both the test

team and the development team and the test team and management. To some extent, this leads to two other cited issues: the test team feels underappreciated and they are unable to illustrate the importance of testing.

2.3.2 TOPICS FOR THE RESEARCH COMMUNITY

Several of the recommended research areas closely align with the most frequently identified software testing problems. The training and education of testers tops both lists, with individuals requesting that work be done on how to train and certify testers in addition to guidelines for tester education.

Equally important for research, respondents indicated a strong need for improving the efficiency and effectiveness of the testing process. This demand ties closely with the need for specific forms of testing, with related requests for research in improved regression testing and more affordable automated and regression testing. A few respondents also cited a need for research in testing without a tester, specifically in agile environments.

While tools received less attention under this question than the software testing problems question, the request for better software testing tools and for assistance in analyzing and selecting tools for specific purposes was highlighted. Similarly, a few individuals referenced the need for better methods of estimation and improved communication between the test team and others.

CHAPTER 3

Testing as a Service (TaaS)

As software systems grow in size and complexity, testing is becoming an increasingly costly endeavor in terms of time, labor, and other resources. Despite the best efforts of software testers, defects continue to elude standard testing techniques, resulting in production environments where errors cost significantly more to repair.

Testing as a Service (TaaS) is a relatively new form of testing that "leverages the vast resources of cloud environments to offer testing services to consumers on an as-needed basis" [2]. TaaS providers tout its potential to revolutionize the industry, promising to greatly reduce efforts while costing less than traditional testing. Nevertheless, TaaS, like any new technology, is not without risks and challenges of its own.

This chapter provides an overview of the current state of TaaS from both academic and commercial standpoints. Chapter 4 compares the capabilities of the current state of TaaS solutions to the testing needs identified by the HPST survey described in Chapter 2.

3.1 KEY COMPONENTS OF TaaS

This section provides background on two key components of TaaS: cloud computing and service-oriented architecture. The cloud provides the test execution environment for the system under test and for the testing tools themselves. Service orientation provides a programmatic interface to the cloud-based testing infrastructure.

3.1.1 CLOUD COMPUTING

According to the National Institute of Standards and Technology (NIST), cloud computing is "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable resources … that can be rapidly provisioned and released with minimal management effort or service provider interaction" [15]. Cloud computing must offer on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service that can be deployed in four separate models:

- The *private cloud* is owned by a single organization that maintains sole access.
- The *public cloud* is available for general consumption.
- The *community cloud* is accessible by a specific set of consumers.

- The *hybrid cloud* is composed of a distinct and independent set of cloud infrastructures (private, community, and/or public).

NIST also defines three models of service, where cloud providers offer a set of underlying infrastructure to be utilized by consumers:

- **Software as a Service (SaaS):** Consumers run applications in the cloud, which are accessible through the web. SaaS providers supply the operating systems, storage, networking, and servers. Examples include Google Docs and Microsoft Office 365.
- **Platform as a Service (PaaS):** Consumers use provided languages, libraries, and tools to deploy applications to the cloud, offering them greater control over applications and their configurations. PaaS providers supply the operating systems, storage, networking, and servers. Examples include Google App Engine and Amazon Web Services (AWS) Elastic Beanstalk.
- **Infrastructure as a Service (IaaS):** Consumers deploy their own operating systems and applications and have access to provider-provisioned processing, storage, networking, and other resources. IaaS providers have some control over the operating systems, storage, and, in some cases, networking. Examples include Amazon EC2 and Windows Azure.

3.1.2 SERVICE-ORIENTED ARCHITECTURE

Service-oriented architecture (SOA) is an architectural style of distributed computing in which functionality is defined by loosely coupled components that are responsible for realizing one or more capabilities, called services [16]. SOA is not restricted to any one technology, but web services are commonly used as a means to provide interoperability. Web services utilize a machine-processable interface written as an XML-based web services description language (WSDL) and communicate with external systems using SOAP messaging, an XML-based protocol containing structured information. TaaS uses services to perform testing responsibilities and report results. Aspects of SOA are mentioned in later sections, particularly in describing TaaS product features in Section 3.3.2.

3.1.3 TESTING AS A SERVICE (TAAS)

Since the NIST's definition of cloud computing was written in 2011, the usage has expanded to include additional types of services with distinctly new purposes. Combining elements of cloud computing and service-oriented architecture, TaaS "offers the ability to test software in the cloud as a competitive and easily accessible web service, allowing tests the ability to harness the vast and elastic resources provided by the cloud infrastructure" [2] [17]. To perform a test, an individual or

automated tool submits the request via a well-defined service. The test is executed per request and all test findings and metrics are returned in the form of a detailed report.

The actual test execution may be handled in three forms:

- The tests can be executed in a *fully automated manner*, based on a set of pre-defined parameters included in the submitted request.
- The tests can be *human-backed*, run in a similar manner to outsourcing where a request is submitted and human intervention in the form of a third party is required to complete the request.
- The tests can be completed through a hybrid mixture of human-intervention and automation.

The goal of TaaS is to offer highly available and highly accessible testing services at a low cost, using a pay-as-you-test pricing model. TaaS aims to provide more thorough testing performed in less time than traditional testing, promising safer and more reliable applications [17]. Various types of testing have already been incorporated into testing services and are described in the following sections.

3.2 ACADEMIC STATE OF TAAS

In academia, various aspects of TaaS have been studied and evaluated. Research into architecture, examination of specific types of testing, and case studies are detailed in the following sections. Benefits, challenges, and needs as discussed by academics are reviewed. Other related research, with the exception of papers overviewing and surveying existing work, is briefly mentioned.

3.2.1 ARCHITECTURE

According to academic research, TaaS utilizes a layered architecture, typically consisting of four to five layers [18] [19]. The actual layers vary to some degree, but most follow the same general flow described by Lian Yu and his associates from Peking University and the IBM China Research Center in 2010 [20]. Figure 3.1 shows Yu's TaaS architecture.

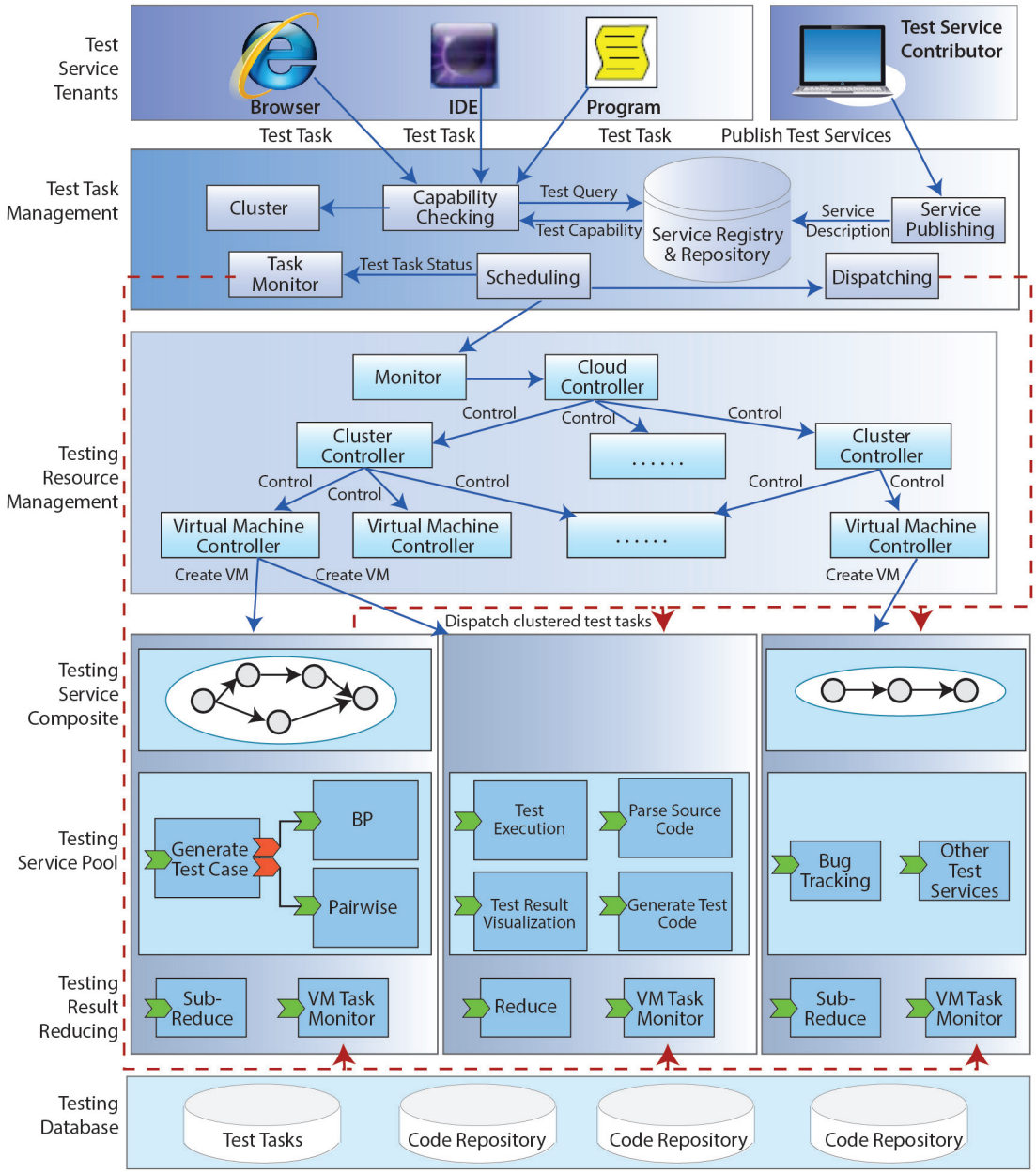


Figure 3.1: Yu's TaaS architecture.

The top layer supports customer interaction, with services being accessible over the internet via various tools and applications as permitted by the service provider. Service requests are submitted by users with any necessary parameters and data. Yu divides the top layer, referring to it as the

Test Tenant and Test Service Contributor layer, where the contributor side allows providers to publish new services directly to the platform.

The remaining layers handle test and resource management, test execution, and, in some cases, a database layer that stores tasks, code, testing services, bugs, and other relevant information. The second layer is the *Test Task Management* layer, which acts as a testing service bus and is responsible for checking capabilities; clustering, scheduling, and dispatching tests; monitoring tasks; and registering, reposing, and publishing services. The *Test Resource Management* layer monitors the physical and virtual resources, allocating them as required. The final layer is the Test Layer, which handles the actual test execution and aggregation of test results. When providers offer test case generation services, the *Test Layer* is also responsible for that functionality.

3.2.2 TYPES OF TESTING

Several academic researchers have focused their efforts on describing or creating testing services that offer the functionality of a specific type of testing. A survey of this research is described in the following subsections.

Predicate and Certification Testing

In 2010, George Candea, Stefan Bucur, and Christian Zamfir of École Polytechnique Fédérale de Lausanne in Switzerland presented three different potential testing services [17], each intended for a unique stakeholder. Their research endeavored to promote TaaS as a powerful tool, permitting complex testing with minimal upfront investments. Each service offered a specific type of testing necessary to guarantee quality to the stakeholder, while demanding little to no knowledge of software testing.

$TaaS_D$ for Developers ("Sidekick") acts as a continuous predicate testing service, automatically pulling the latest code from the source code repository to test against a series of vendor- and developer-defined predicates, described respectively as universal and application-specific predicates. The service walks through execution paths, creating assertions based on the predicates. Results are returned to the developers to promote better software development. $TaaS_D$ should be run prior to submitting code for test.

$TaaS_H$ for End Users ("Home Edition") is a separate predicate testing service intended for end users that want to verify the quality of a software application on their personal system with minimal effort. Users simply upload the application in binary or bytecode form and the service examines it for common bugs, such as buffer overflows, deadlocks, and race conditions. The service returns a webpage containing all results in a readable and easily understood format, including any identified issues ranked by severity. $TaaS_H$ is less flexible than $TaaS_D$, but offers quality assurance to stakeholders with no knowledge of software testing.

$TaaS_C$ Certification Services is a public certification service that provides an objective assessment and rating of the quality of an application under test. Based on the funding government or industry, the application can be tested and certified on varying levels, differing in the predicates against which it is examined. A complete report is automatically generated for internal use or for publishing statistics on the software's quality and reliability. $TaaS_C$ can be rerun with each subsequent release, quickly recertifying the system.

Optimized Regression Testing

Sheng Huang and his associates from Fudan University, Peking University, and IBM China Research Lab described a selective regression testing platform called RTaaS in 2011 [21]. RTaaS utilizes their optimized regression test selection tool, ORTS, to seek out the minimal set of regression tests affected by new or modified code, submitted in the form of a project binary. Impacted regression tests are then executed in the cloud environment, saving time and effort by reducing the number of tests being rerun and by taking advantage of the cloud's resources. RTaaS's architecture is shown in Figure 3.2. Experiments on the RTaaS platform are described in the section on academic case studies.

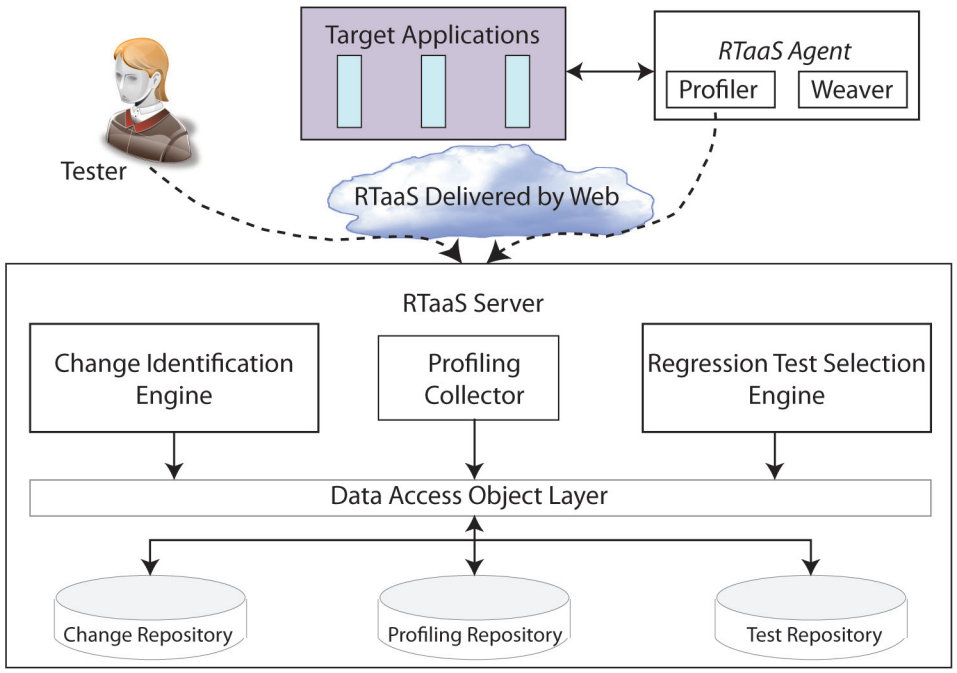


Figure 3.2: RTaaS architecture.

Web Service Load Testing

In 2012, Minzhi Yan and fellow researchers from Beihang University in Beijing, China, identified challenges in pre-existing web service load testing environments and laid out four requirements for a better, TaaS-based system [22]. Their WS-TaaS implementation, based on their Service4All PaaS platform, needed the following:

- Transparency, negating the tester's need for knowledge about the hardware and software configuration;
- Elasticity, allowing the test capabilities to scale appropriately with demand;
- Geographical distribution, simulating real world runtime scenarios of user access from varying locations across the globe; and
- Massive concurrency and sufficient bandwidth, offering the ability to manage wide test loads concurrently and with sufficient bandwidth to complete the request.

Yan's WS-TaaS layered architecture, similar to Yu's in execution, is shown in Figure 3.3.

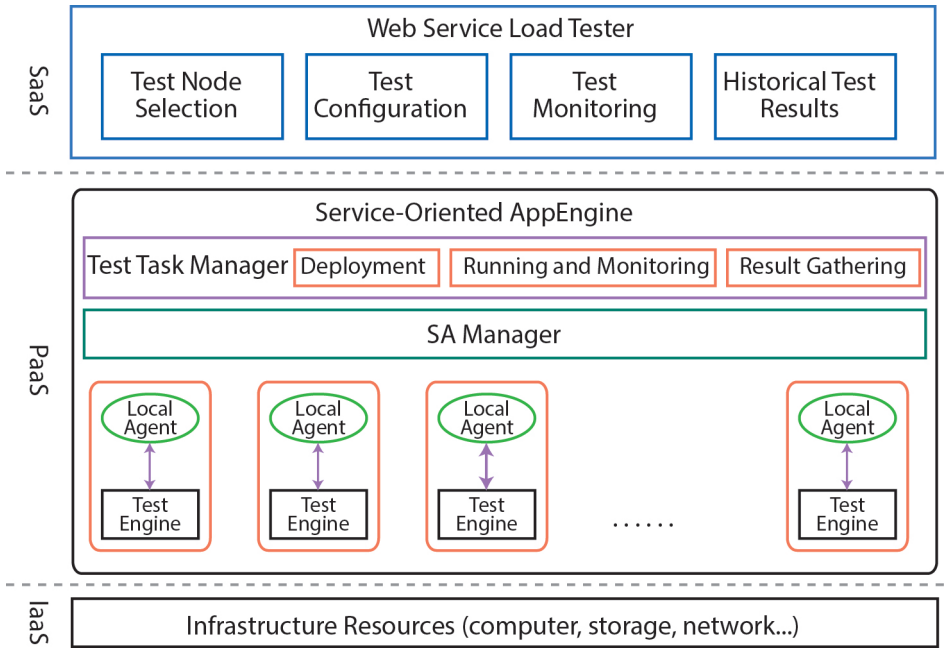


Figure 3.3: WS-TaaS architecture.

WS-TaaS has three different test modes [23]. A static test examines the performance of the web service using a pre-defined load parameterized by the user. A step test studies the web service's ability to cope with a specific load span, beginning with a start number, step size, and end number of concurrent requests. A maximal test determines the web service's load limit, stressing the service until more than 10% of the concurrent requests fail. Various experiments were performed against WS-TaaS and are described in the academic case studies section.

Structural Unit Testing

In 2013, Nikolai Kosmatov and his colleagues from the CEA LIST Institute's Software Reliability Laboratory in Gif-sur-Yvette, France, discussed their work on structural unit testing with Path-Crawler-online.com and its potential as a TaaS application [24]. PathCrawler is an online service that accepts source code written in C and uses concolic or dynamic symbolic execution to generate test case inputs that guarantee structural coverage of the application. PathCrawler is not currently deployed in the cloud for test execution, but is actively under development and has been considered a good candidate for becoming a TaaS application. The authors describe the need for unit testing in the cloud and the potential role that PathCrawler could play as a developer's sidekick.

PathCrawler works by compiling the submitted C source code into an executable test harness, called a Launcher. The Launcher's executable paths are explored by the Path Explorer, which sends partial paths to the finite-domain constraint Solver. The Solver returns the test inputs for that path to the Launcher, which can then execute the system under test with the data. The process is cyclic, allowing the Path Explorer to continue down the next partial path. When the Solver is no longer able to generate a test case or a timeout occurs, the Path Explorer seeks an alternate path. This process is illustrated in Figure 3.4. When complete, PathCrawler generates and returns all test cases, along with coverage and path statistics.

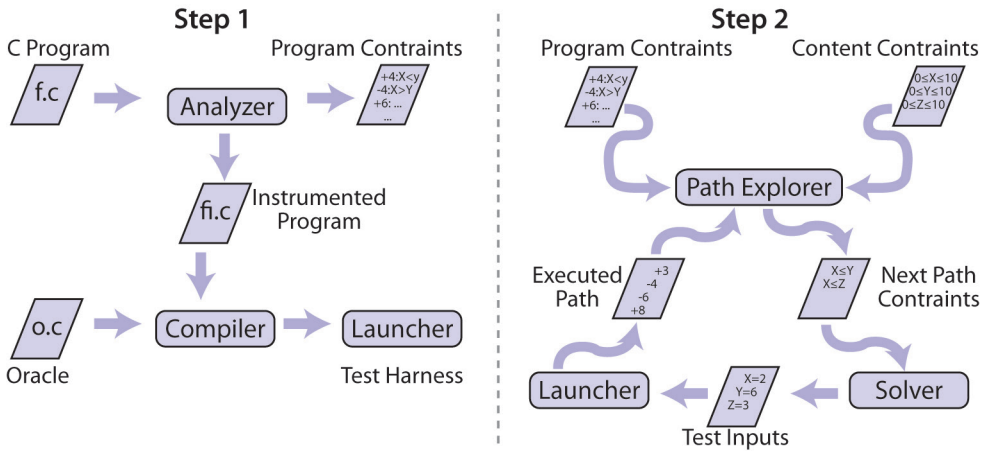


Figure 3.4: PathCrawler method.

3.2.3 CASE STUDIES AND EXPERIMENTS

Many of the academic researchers conducted case studies and experiments with their TaaS systems. The results provide useful insight into the state-of-the-art in the area.

Performance of Local and Remote Systems

Yu and his colleagues built their TaaS platform on Apache Tuscany using Java and C++ [18]. They selected two white box and two black box services for the generation of test cases: BP (Basic Path), BRO (Branch and Relational Operator), GEO (Generalized Extremal Optimization), and Pairwise. Their first test evaluated the difference in service invocation times between local and remote systems using all four services. Figure 3.5 shows that an additional 2.5–3.5 seconds was required for remote invocation. The increase was attributed to the necessary time required to communicate between test tenants and the TaaS platform.

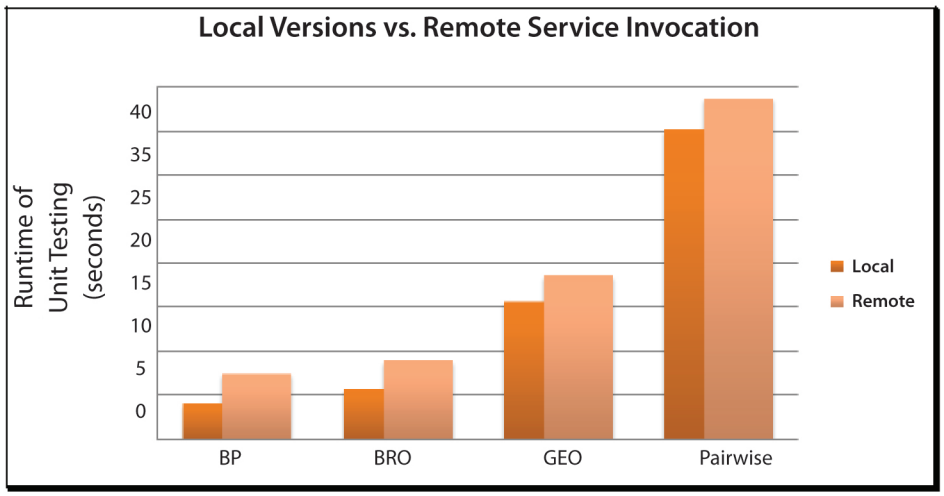


Figure 3.5: Local vs. remote invocation times.

Their second experiment used BP and BRO and demonstrated the performance of test case generation while increasing the size of the system under test, taking the average time from ten trials at each size. Figure 3.6 shows the runtime when increasing the lines of code in the application; Figure 3.7 shows the runtime when increasing the number of conditional branches in the application. When increasing the lines of code, additional time was only necessary for parsing the source. On the other hand, the tools were sensitive to increases in complexity. Significant increases

in time were required when the number of conditional branches increased beyond twenty, but was considered acceptable otherwise.

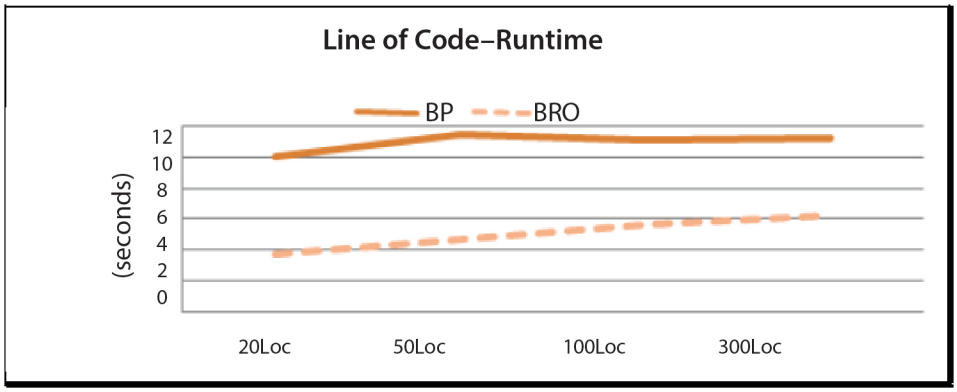


Figure 3.6: Runtime with increasing lines of code.

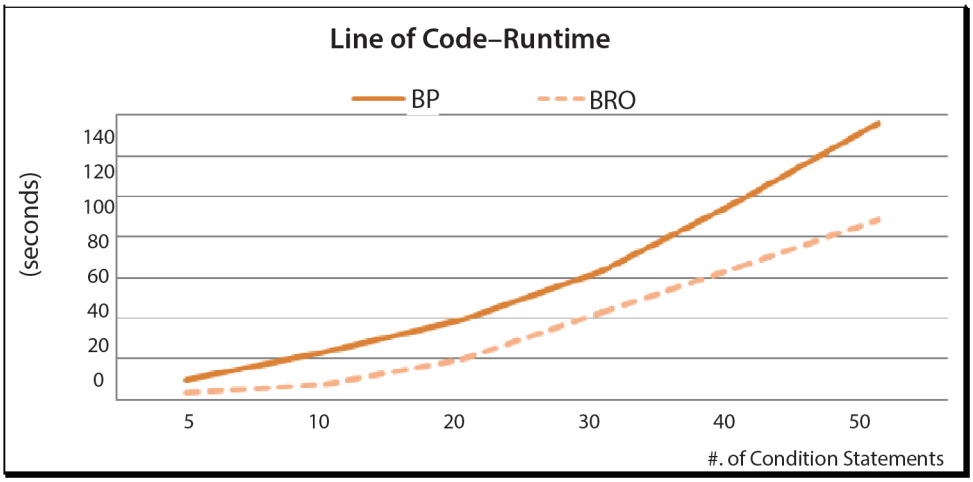


Figure 3.7: Runtime with increasing number of conditional branches.

Experiments with RTaaS

Huang et al. deployed their RTaaS tool to a software park in China for study [21]. Three groups participated in the case study. Project A was a small project under IBM China that required only minor function updates or occasional bug fixes. Projects B and C were both under development by

independent software vendors that were struggling with the time required for regression testing. Table 3.1 offers the basic information on each project, including number of files, source lines of code (in thousands), programming artifacts, test iterations, and test cases. Programming artifacts include any external files required for execution, including software libraries (Java, JSP, JavaScript) and configuration files. For the purpose of this case study, all projects used comparisons between the optimized regression test selection and rerunning all regression tests.

Table 3.1: Project information

Project	Files	KLOC	Artifacts	Iterations	Test Cases
A	1,642	1,960	14,220	1	600
B	162	38	1,227	4	50
C	472	185	2,705	3	194

The regression test selection (RTS) time was studied across all projects. As package unzipping occurred offline, only the change identification and test selection were considered in the RTS time. The results are shown in Figure 3.8.

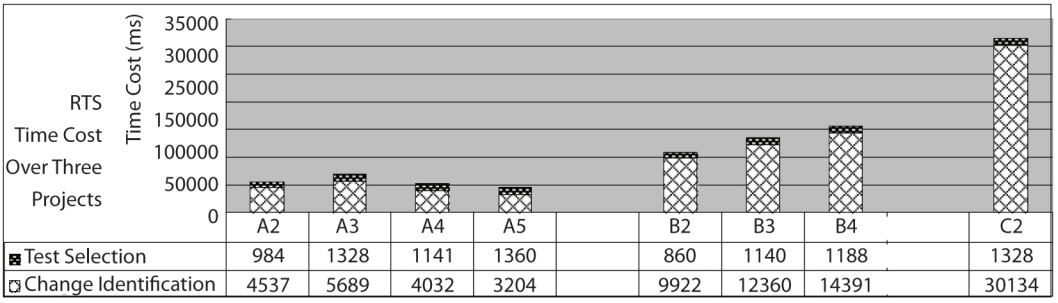


Figure 3.8: RTS cost across project iterations.

Additionally, Huang and his associates examined the effectiveness of the selection strategy, which they referred to as its safety. Rerunning the entire regression test suite is considered very safe, but costs too much in terms of time consumed. In each project iteration, the rerun all and RTS methods found the same number of defects, suggesting they shared the same high safety for these cases. The results of this experiment are shown in Figure 3.9.

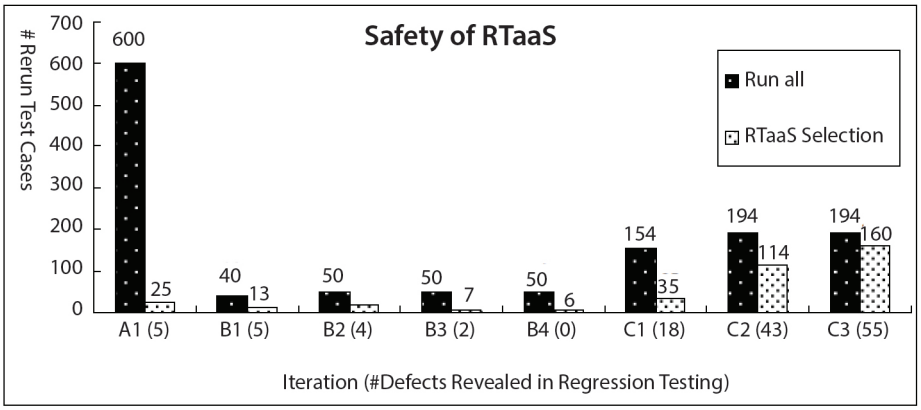


Figure 3.9: Safety of RTaaS.

Experiments with WS-TaaS

Yan et al. performed experiments to compare the performance of WS-TaaS against traditional single node JMeter load tests [23]. Their first test compared the average response time (ART) of three services under varying loads. Figure 3.10 shows that the ART of the JMeter tests increased significantly with an increasing number of concurrent requests, while WS-TaaS tests' ART remained relatively consistent despite load increases.

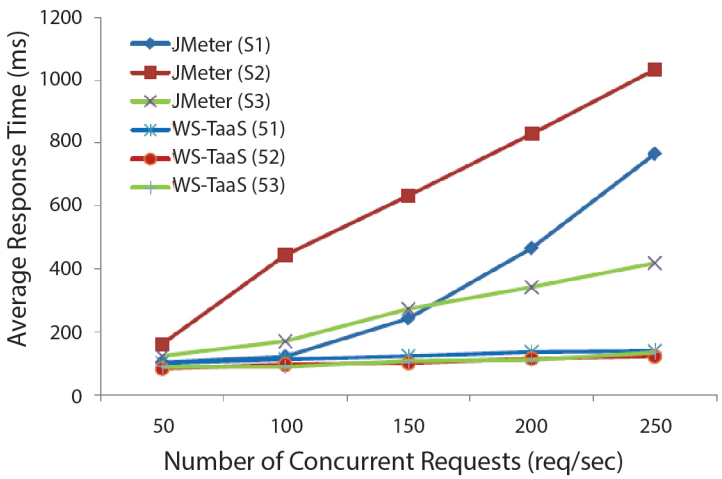


Figure 3.10: ART comparison with increasing load.

Another test analyzed the error ratio of JMeter and WS-TaaS when running load tests against a web service that validates email addresses. As shown in Figure 3.11, the load limit for WS-TaaS was significantly less than JMeter.

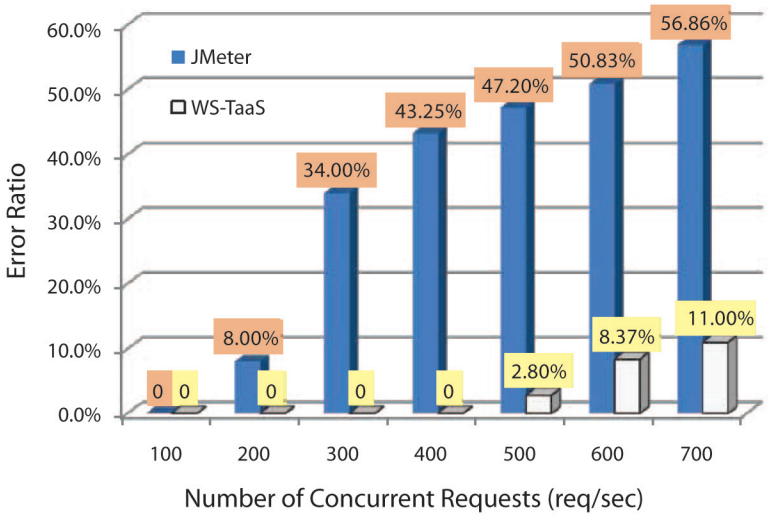


Figure 3.11: Error ratio over increasing loads.

3.2.4 BENEFITS, CHALLENGES, AND NEEDS

A significant amount of early research into TaaS focused on its potential, exploring anticipated benefits and migration feasibility, and its risks, seeking challenges that must be addressed. These items are discussed below.

Benefits

One of the most commonly cited benefits of TaaS in academic literature is reduced cost. Software development has become a costly endeavor when considering employee salaries and benefits, physical building space and utilities, equipment upgrades and maintenance, and software licenses [2]. Each of these assists in driving the cost of the final product up, which must then enter a competitive market where prices frequently affect success. TaaS promises an overall reduction in the costs associated with testing by offering testing services at a pay-as-you-test rate. The base rate includes a large variety of environments and hardware configurations with licensed software, including operating systems, preconfigured and available for use. Not only does this reduce the cost of test

lab machines, maintenance, software, and space—TaaS reduces the amount of overall labor cost by minimizing the number of testers required for the project. With access to the cloud's vast and elastic resources, virtualized servers and on-demand systems are offered at a price point that is simply unachievable with traditional test labs [13].

Another major benefit of TaaS is a more efficient and less time-consuming testing process. In traditional testing, large test suites may take hours or days to run, sometimes on just one of the several environments under test. TaaS offers the option to run these tests with more resources and in more environments and configurations simultaneously, thereby eliminating the restrictions imposed by traditional testing's physical labs and tight schedules. The services handle all of the resource provisioning, reducing setup time and allowing for a more fluid testing process. Results are returned to the testers after test completion and can be rerun as necessary, providing the continuous regression testing availability necessary for agile development [11]. With much greater access to environments and configurations, testers have a better indicator of the system under test's quality and reliability.

Additionally, the testing services are highly consumable, allowing testers to quickly learn the TaaS tool with minimal effort; the complicated aspects of testing are handled by the provider, out of the tester's sight. In Huang's case study, users mastered the basic steps of the RTaaS tool within hours [21].

Challenges

As expected with any new technology, the benefits are balanced by issues and challenges. Perhaps the most significant challenge is maintaining constant global availability. Due to the nature of software development, test services must be accessible by end users at all times, providing flexibility and enhancing agility in testing [13]. While maintaining availability may be a difficult task on the provider's side, the failure to do so could have crippling consequences on a consumer by causing lost time and data. Additionally, customer and technical support must be equally available to assist testers, especially when working in international markets. These staff members must maintain skills that match or exceed the needs of their clients to provide the best customer experience.

Another major challenge that exists across online services is security. Consumers must be confident that their proprietary information is secure and that any system flaws remain confidential. In addition to application security, any issues with storage or mismanagement of test data could compromise a system [13]. One option is to provide mock data that excludes sensitive information. At the same time, adequate quality measures sometimes depend on real production data [11]. Nondisclosure agreements may help mitigate some of these security concerns.

Existing software components that lack a standardized API for the cloud infrastructure may have connectivity problems, incurring increased integration costs [25]. Any interoperability challenges between the system under test and the cloud or TaaS tool could have disastrous effects on

the testing process. Interoperability risks and other unsatisfied non-functional requirements could dissuade potential customers from adopting TaaS technologies.

As affordable testing services become available to the public, end users with less benevolent intentions could adopt TaaS tools to locate and exploit weaknesses in software. Candea et al. acknowledge these concerns, but believe that they may compel software companies to perform additional testing to mitigate risks and release better quality products [17].

Needs

Beyond the aforementioned challenges, TaaS has various needs that must be satisfied. Pricing models must be transparent and descriptive to educate customers, allowing them to make informed estimations of their expenses [12]. The description of services must be equally compelling, providing detailed information on service definitions and reportable metrics.

Standards for TaaS must also be created. Both the consumers and providers must have well-defined service standards that illustrate testing and quality details to assist in the development of fair Service Level Agreements (SLAs) [25]. At the same time, standards must not be overly restrictive, as that could lead to more predictable testing [11].

TaaS cannot succeed without innovative test methods and solutions, both to increase demand and to maintain its relevance in the future. Additionally, dynamic test platforms and tools must be used to provide powerful test simulators, develop flexible interfaces for existing tools and solutions, offer seamless integration, validate scalability and performance, and more [25].

Finally, as stated by Riungu-Kalliosaari et al., "cloud-based testing is more efficient and effective than traditional testing methods but organizations must understand and trust it before it sees widespread adoption" [13]. TaaS must prove itself to the community prior to being fully utilized.

3.2.5 OTHER RESEARCH

There are a few other areas of research that can affect the future of TaaS. These include mobile testing platforms, knowledge management systems, and a new type of fault injection testing technique called "failure as a service."

Mobile TaaS Framework

Oleksii Starov and Sergiy Vilkomir of East Carolina University proposed a mobile TaaS platform called Cloud Testing of Mobile Systems (CTOMS) [26]. In 2013, they published a paper discussing the system, which consists of a cloud of devices, a static analysis engine, and a statistics sub-system. Testers can select from this pool of mobile devices for functional, performance, or manual testing. In their research, Android was the focal platform due to its widespread use and open-source platform.

CTOMS was developed as a master application deployed to the cloud with slave servers acting as nodes. The master application is responsible for organizing the core functionality of the system by collecting information, organizing and distributing work, and gathering the results for users. The nodes perform the actual testing, manage the devices, and collect statistics. The implementation is depicted in Figure 3.12.

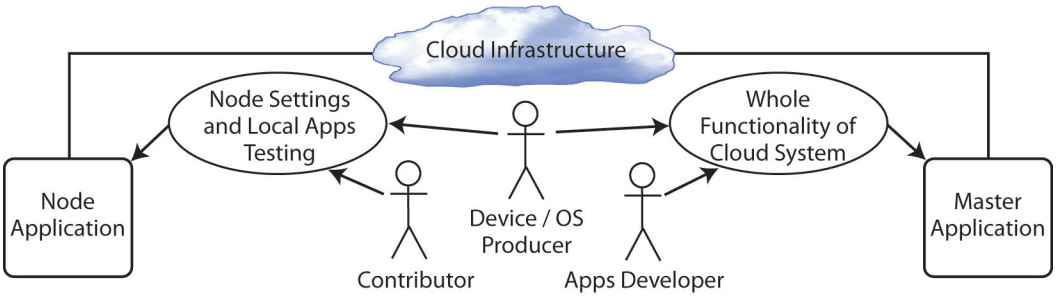


Figure 3.12: CTOMS implementation.

Knowledge Management System

Rusli Abdullah of the University Putra Malaysia described the need for a Collaborative Knowledge Management System (CKMS) for TaaS [27]. His work rightly promotes the necessity of disseminating acquired knowledge to the community to assist in an overall reduction in errors and an improvement in quality of TaaS systems. The repository of knowledge would be structured as detailed in Figure 3.13. The repository is generic enough to be useful for many situations beyond just TaaS knowledge.

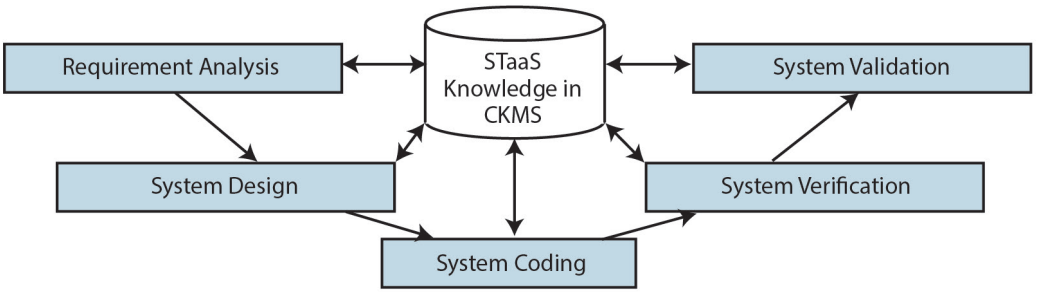


Figure 3.13: Structure of CKMS Repository.

Failure as a Service

Haryadi Gunawi (now at the University of Chicago) and his colleagues at the University of California, Berkeley, researched another aspect of system quality. Their work proposed a new cloud service that permits routine large-scale failure drills, allowing testers to pinpoint problems preventing a normal recovery following a service outage [28]. In many high profile cloud software outages, the issues were prolonged due to a failure to consider all potential complications after the system went temporarily offline. While not directly related to TaaS, Gunawi et al.'s Failure as a Service (FaaS) tool offers an additional means to verify quality in a software vendor's cloud application.

3.3 COMMERCIAL STATE OF TAAS

Numerous companies are already providing TaaS tools and solutions. Twelve different companies are discussed in this section. Of them, three are discussed in much greater detail during the case study provided in the following chapter. Table 3.2 lists the vendors currently involved in TaaS along with their website. The company name is located in the first column and is emphasized with bold characters. If the product name differs, it is listed below the company name in the normal font.

Table 3.2: TaaS vendors

Company and Product	Website
Applause (Formerly uTest)	http://www.utest.com
Appvance (Formerly PushToTest)	http://www.appvance.com
BlazeMeter	http://www.blazemeter.com
Bulldog QA	http://thebulldogcompanies.com
CA (Formerly ITKO) LISA	http://www.itko.com
Cigital	http://www.cigital.com
CSC	http://www.csc.com
HP LoadRunner OnDemand	http://www.hp.com
IBM Smart Business Test Cloud	http://www.ibm.com
Oracle	http://www.oracle.com
Sauce Labs Sauce OnDemand	http://www.saucelabs.com
SOASTA CloudTest, CloudTestLite, TouchTest	http://www.soasta.com

As TaaS is a relatively new aspect of computing, commercial offerings are subject to change. Furthermore, it is important to note that the information contained in the remainder of this chapter is based on advertised features that were not, in the majority of cases, validated during the course of this research for availability, functionality, or quality. Many tools, especially the larger commercial products by CA, HP, IBM, and Oracle, were not available for trial or limited use. Additionally, absent information may not indicate that a feature is not provided; rather, the feature simply is not advertised by the vendor.

3.3.1 TYPES OF TESTING

The aforementioned companies deliver a wide variety of testing services. An overview of comparable services is displayed in Table 3.3. The forms of testing offered by TaaS vendors are defined below. Unless otherwise specified, the definitions are based on IEEE Standard 61012-1990 [29] and/or ISO/IEC/IEEE 29119-1 [30, 31, 32].

- Concurrency Testing: A form of testing that evaluates the behavior of two or more components when run within the same time interval.
- Domain Testing: A form of testing focused on systematically covering different input situations based on the input domain, negating the need to exhaustively cover all possible input values [33].
- Functional Testing: A form of testing that ignores the internal workings of the component or system and instead evaluates the outputs generated in response to inputs and execution conditions.
- Infrastructure Testing: A method of testing the system's underlying infrastructure.
- Integration Testing: A form of testing that evaluates the interaction between software and hardware components.
- Load Testing: A form of performance efficiency testing used to evaluate how a system behaves under conditions of varying load, as anticipated during low, typical, and peak usage.
- Localization Testing: A type of testing that verifies the quality of a system's translation of its interface for suitability in another region [34].
- Performance Testing: A form of testing that evaluates the ability of a system or component to accomplish designated functions with time and/or resource restrictions.

- Regression Testing: A method of selectively re-testing previously evaluated behaviors of a system or component following modifications to ensure that it still complies with the original requirements and that the change had no unintended consequences.
- Security Testing: A form of testing that evaluates the ability of a system to protect a test item and its associated data from unauthorized persons or systems while allowing access for authorized persons or systems.
- Stress Testing: A variation on load testing that evaluates how a system behaves under conditions of higher than anticipated or specified load or with below specified resource availability.
- Unit Testing: A form of testing that scrutinizes the behavior of individual or related software units.
- Usability Testing: A form of testing that judges how easily a system or component can be understood and operated by a user.

Table 3.3: Types of testing available

Company	Functional	Load	Performance	Security	Unit
Applause	✓	✓		✓	
Appvance		✓	✓	✓	
BlazeMeter		✓	✓		
Bulldog QA		✓	✓		
CA LISA	✓	✓			✓
Cigital				✓	
CSC	✓	✓			
HP	✓		✓	✓	
IBM	✓		✓	✓	
Oracle	✓	✓			
Sauce Labs					✓
SOASTA	✓	✓	✓		

In addition to the types of testing listed in the above table, several vendors have unique services available to consumers. Some of these unique services include:

- Applause: Localization and Usability Testing
- BlazeMeter: Stress Testing
- CA LISA: Concurrency and Regression Testing

- CSC: Domain Testing
- IBM: Infrastructure and Integration Testing

3.3.2 PRODUCT FEATURES

The advertised product features of each tool are displayed in the next three tables. Table 3.4 shows the standardized protocols and system types evaluated by each tool. Table 3.5 lists the supported test script languages and notes whether tests can be recorded and played back rather than written from scratch. Table 3.6 catalogs compatible tools for version control, driving builds, continuous integration, and defect tracking. Unfortunately, several of the vendors offer little information on their tools and services, which accounts for many of the blank cells in the following tables.

The following describes the protocols that are employed by vendors. Several of these web service descriptions are based on CMU/SEI-2010-TR-011 [35]:

- HTTP (Hypertext Transfer Protocol): Services are tested for conformance over HTTP.
- HTTPS (HTTP Secure): Same as HTTP, but the services require verification for security, such as checking for the validity of an SSL certificate.
- XML-RPC (Extensible Markup Language-Remote Procedure Call): An XML-based protocol that transfers information over HTTP. XML-RPC proceeded SOAP.
- SOAP (Simple Object Access Protocol): An XML-structured protocol for exchanging information in web services. Can be used in HTTP and non-HTTP situations, such as TCP and named pipes.
- REST (Representational State Transfer): A more recent and lightweight standard that utilizes HTTP to transfer information in web services.
- WSDL (Web Services Description Language): An XML-based description of a web service, often used in combination with SOAP to provide web services.

Additionally, databases and applications are included in Table 3.4 and are used in the following context:

- Databases (DB): The database information can be accessed for verification via SQL (MSSQL, MySQL, Oracle, etc.).
- Applications (Apps): The types of applications that are testable by the tool.

Table 3.4: Supported protocols and systems

Company	HTTP/S	SOAP	XML	REST	WSDL	DB	Apps
Applause							Web, Mobile, Desktop
Appvance		✓		✓			Web
BlazeMeter	✓					✓	Web, Mobile
Bulldog QA							
CA LISA		✓	✓		✓	✓	Web
Cigital							
CSC							
HP							
IBM							
Oracle							
Sauce Labs	Sauce Connect			✓			Web, Mobile, Native
SOASTA	✓	✓		✓	✓		Web, Mobile, Native

Table 3.5: Supported test script languages

Company	Recording	Test Script Languages
Applause		
Appvance	✓	soapUI, Selenium, Sahi, JUnit, Ruby, PHP, Perl, Java
BlazeMeter	✓	JMeter
Bulldog QA		
CA LISA	✓	Java, .NET
Cigital		
CSC		
HP		
IBM		
Oracle		
Sauce Labs	✓	Selenium, JavaScript, Appium, Java, Python, Ruby, Node JS, PHP, C#.NET
SOASTA	✓	JMeter

Table 3.6: Compatibility with external tools

Company	Version Control Systems	Drivers	Continuous Integration Systems	Defect Tracking Systems
Applause				
Appvance	Git		Jenkins, Hudson, Bamboo	
BlazeMeter			Jenkins, Bamboo, TeamCity	
Bulldog QA				
CA LISA				
Cigital				
CSC				
HP			Jenkins	
IBM				
Oracle				
Sauce Labs	GitHub	Maven	Jenkins, Travis, Bamboo	
SOASTA			Jenkins, Hudson	

Note: By integrating with Jenkins CI, consumers gain access to the Jenkins plugin library. This library consists of version control systems, drivers, defect tracking tools, and more.

CHAPTER 4

Case Study and Gap Analysis

This chapter presents a case study that provides the basis for an evaluation of TaaS against the issues identified in the HPST survey. The evaluation framework is described and applied to the tools examined in the case study. Of the twelve tools introduced in the previous chapter, three were chosen as candidates for additional analysis: Sauce Labs, SOASTA CloudTest Lite, and BlazeMeter. The remainder of this chapter details the case study and provides an analysis of the gap between current TaaS solutions and industry needs.

4.1 TaaS TOOLS

From the summary tables in Chapter 3, the unavailability of detailed information on some the TaaS products becomes quite obvious. Of the companies that offered significant information about their product, only three promoted free, albeit limited, versions of their tools to potential consumers as a means to learn the tool and determine if it suits their needs. Surprisingly, very few companies offered trial periods with their products. More commonly, interested parties can request a demonstration of the tool as a means of better understanding its features and uses.

For the purpose of this case study, all three of the freely available tools were examined. Each is described in greater detail below, before being explored as part of the case study in Section 4.2. The descriptions in this section are based on the advertised features from the provider's educational material and, to remain objective, have not yet been validated for functionality, quality, or usability. This section simply aims to offer a better understanding of the capabilities of each tool, building on the brief overview in Chapter 3.

4.1.1 SAUCE LABS (SAUCE ONDEMAND)

Sauce Labs [36] is a TaaS platform specializing in unit testing. Sauce Labs, sometimes referred to as Sauce OnDemand, became available in 2008. They offer the ability to test web applications using Selenium and JavaScript, and native and hybrid mobile applications using Appium. Additionally, Sauce Labs allows consumers to perform manual tests on over two hundred desktop and mobile platforms, reducing the need for in-house test labs for the purposes of manual testing and user acceptance testing. The available testing environments are listed in Table 4.1, where the numbers in the cells indicate the supported versions of the browser. The mobile environments (iOS and Android) have the same support for both phone and tablet versions.

Table 4.1: Sauce Labs supported environments

	iOS	Android	IE	Firefox	Chrome	Safari	Opera	Lynx
iOS	4-7							
Android		4						
Windows 8.1			11	3-26	26-31			
Windows 8			10	3-26	26-31			
Windows 7			8-10	3-26	26-31	5	11-12	
Windows XP			6-8	3-26	26-31	5	11-12	
OS X 10.6				4-26	27-28, 31	5		
OS X 10.8					27-28, 31	6		
OS X 10.9				4-26	31	7		
Linux				3-26	26-30		12	2

Sauce Labs has a variety of packages to meet the needs of different teams. The packages vary in the number of testing minutes, the number of parallelized tests that can be run concurrently, and the number of distinct user accounts. The paid packages are shown in Figure 4.1 below. Subscriptions can be changed at any time, with new plans taking effect at the beginning of the next billing cycle. Some plans offer rollover minutes. Overages are billed at an additional $0.05 per minute for Windows, Linux, and Android or $0.13 per minute for Mac and iOS.

The free version, which was used in the case study, offers 30 manual minutes, 100 Windows/Linux/Android minutes, 40 Mac/iOS minutes, two parallelizations, and one user account, with minutes being per month. Each minute is one minute of real-time testing—if a test takes three minutes to run, three minutes are deducted from the user's account. Sauce Labs provides bonus minutes to accounts, offering 50 additional minutes after the first test is run and 1,000 additional minutes in celebration of 50 automated test executions. Another free version is available to open source projects. Open Sauce allows unlimited public tests, but must be connected to a project stored in GitHub.

Sauce Labs uses Sauce Connect, a custom HTTP/S-like protocol that uses SSL for securely testing applications behind a firewall. They promise pristine virtual machines that are dismantled after every test to keep all tests and test data secure. During test runtime, Sauce Labs records screenshots, videos, and HTML logs to help with debugging and additional verifications. Additionally, sessions and videos can be shared amongst team members for better collaboration in the Small Team and Enterprise packages.

Users do not have to be skilled in writing automated tests to make use of Sauce Labs. The open source tool Selenium Builder [37] can directly connect to a user's Sauce Labs account, allowing the recording of tests in Firefox using the Selenium Builder plugin. Tests are recorded in Selenium 1 or 2 individually or as test suites and can be run directly in Sauce OnDemand from

the plugin. Alternatively, tests can also be exported as a Java, JUnit, or TestNG file for editing or use with other tools.

Figure 4.1: Sauce Labs pricing.

Continuous integration is offered through external tools, including Jenkins, Bamboo, and Travis, with Hudson coming soon. Apache Maven is fully supported, allowing tests to be run on Sauce Labs' platforms during the standard build process. Additionally, open source projects can be directly connected to an Open Sauce account.

4.1.2 SOASTA CloudTest LITE

SOASTA [38] was founded in 2006 to provide website and web application testing services. They offer a variety of tools including CloudTest and its free but limited counterpart CloudTest Lite. CloudTest is a functional, load, and performance testing tool built on Amazon Elastic Compute Cloud (EC2) that tests websites and web and mobile applications with global traffic. CloudTest can execute tests with up to millions of users and thousands of servers; however, CloudTest Lite is limited to a single server and up to 100 virtual users. Another tool, TouchTest, is packaged with CloudTest Lite and allows precise capturing of multi-touch gestures for playback on mobile devices, realistically testing iOS and Android applications.

According to SOASTA's educational material, CloudTest Lite is intended for implementing tests early in the development cycle, working in a small-scale environment, verifying web applica-

tion functionality, and allowing prospective customers to freely evaluate CloudTest's basic capabilities. While larger programs should use the full version of CloudTest, pricing information is not available on the website.

Test composition, execution, and monitoring are all handled in one platform. Existing Apache JMeter tests can be imported directly into the SOASTA platform; alternatively, test scripts can be recorded through browsers, including Firefox and Chrome, using Windows, Mac, and Linux systems. Recorded test scripts are converted into test clips, where they can be edited to include data verifications. Clips can be combined and run concurrently or consecutively as test compositions. At runtime, results and data become immediately available.

SOASTA includes real-time analytics with all tools. Their custom OLAP engine provides the detailed data necessary to allow testers to explore live results during test execution, pinpointing bottlenecks and areas of stress within the web application. The system includes a built-in, end-to-end view of web performance, consolidated in charts, lists, and graphs. Performance data includes database servers, load balances, code, bandwidth, and end user response time.

In conjunction with CloudBees, a Java-based PaaS, SOASTA incorporated Jenkins into their offerings, allowing users to build, test, and deploy applications. Hudson can also be leveraged for continuous integration purposes.

4.1.3 BlazeMeter

BlazeMeter [39] is a load, performance, and stress testing tool also built on Amazon EC2 that allows users to simulate scenarios on websites, web and mobile applications, and web services. The tool is fully compatible with Apache JMeter, "the gold standard in open source performance testing," allowing users to import entire suites of existing tests. BlazeMeter is highly scalable, allowing for over 300,000 concurrent users with up to 100 dedicated servers with geographically distributed loads.

BlazeMeter was founded by Alon Grimonsky in response to developers who demanded a cheaper and simpler alternative to load testing tools like HP LoadRunner and Gomez, which they claimed needed professional service engagements to properly deploy and maintain [40]. Grimonsky raised $1.2 million in venture funding and created BlazeMeter, which would accept the developers' preexisting JMeter tests while offering more powerful and sophisticated tests for advanced web applications, competing against the more comprehensive solutions, including SOASTA. Grimonsky claims that BlazeMeter is the only tool capable of handling "complex testing simulations, unlimited testing capacity, interactive real-time reporting, and sophisticated result analysis and recommendations" [40].

In signing up for BlazeMeter, users automatically receive the free tier of service. The free account offers up to ten tests per month. Each test can, at maximum, extend for an hour's time and have up to 50 concurrent users. The paid packages vary in maximum number of users, load servers,

test hours per month, and maximum test duration. The monthly subscriptions also offer rollover minutes and may include unlimited sandbox testing, master/slave tests, multiple user accounts, and access behind a corporate firewall. Customers only interested in running a handful of tests can instead pay per test. Figure 4.2 shows BlazeMeter's pricing plans.

	Basic	Pro	Pro Plus	Hi Volume	Enterprise
	$199 Per Month	$499 Per Month	$999 Per Month	$1499 Per Month	Call us 1-855-445-2285
Max Concurrent Users	1,000	5,000	10,000	40,000	100,000
Max Load Servers (IPs)	1	5	10	40	100
Test Hours per Month	20	25	25	30	Unlimited
Maximum Test Duration	1	4	6	8	Unlimited
Test Hour Rollover	✓	✓	✓	✓	✓
Unlimited Sandbox Testing		✓	✓	✓	✓
Master / Slave Tests		✓	✓	✓	✓
Multiple User Accounts					✓
Behind Firewall Access					✓
Support Level	Email	Email	Email	Email	Phone

	Basic	Pro	Pro Plus	Hi Volume
	$19 Per Test (5 test minimum)	$49 Per Test (5 test minimum)	$149 Per Test (3 test minimum)	$299 Per Test (2 test minimum)
Max Concurrent Users	1,000	5,000	10,000	40,000
Max Load Servers (IPs)	1	5	10	40
Maximum Test Duration	1 hour	1 hour	1 hour	1 hour
Support Level	Email	Email	Email	Email

Figure 4.2: BlazeMeter pricing.

Users can upload existing JMeter scripts or record them using BlazeMeter's own Chrome plugin. Tests recorded in the plugin may be executed directly or exported, edited, and uploaded into BlazeMeter at a later time. If desired, URLs can be tested directly with GET and POST requests. Users of Google Analytics, a tool that generates traffic statistics, can connect their accounts, allowing BlazeMeter to extract the website's data directly and handle test case generation based on historical data. Finally, companies using Drupal, an open-source content management framework, can download the BlazeMeter's Drupal module, specify the load, and run a performance test without scripting.

BlazeMeter provides a real-time interactive test monitoring dashboard with detailed charts and graphs. The Application Performance Monitoring (APM) system aims to provide detailed application performance to help pinpoint and analyze bottlenecks. Additionally, the APM can integrate with other advanced monitoring tools, including New Relic, to provide end-to-end visibility.

BlazeMeter also works with several continuous integration tools, including Jenkins, Bamboo, and TeamCity.

4.2 CASE STUDY

This section presents a case study that analyzes Sauce Labs, SOASTA CloudTest Lite, and BlazeMeter against an evaluation framework developed using the top challenges identified over the course of the HPST project. A limited selection of features from these tools is applied to a Java web application built on Google App Engine. The following sections describe the case study in more detail, including objectives, the evaluation framework, the system under test, the exploration of the tools, and an overview of results.

4.2.1 OBJECTIVES

The purpose of this case study is to evaluate TaaS tools against the needs of the industry as identified in the HPST survey. As a reminder, the top five challenges are listed below, but additional information can be found in Chapter 2. These points are used to generate the hypothesis that is available in the next section:

1. Tester education and training
2. Need for tools or need for better tools
3. Lacking, insufficient, or non-existent testing (particularly in the areas of automated, regression, and unit testing)
4. Generating realistic schedules that partition out sufficient time for testing
5. Improving communication between the test team and other groups

4.2.2 HYPOTHESIS

Using the aforementioned list of industry needs, five hypotheses were posited. In order to assist in determining whether or not the TaaS tools address the challenges, additional criteria were written based on standard tool evaluation questions and additional related problems discussed in the survey results. The hypotheses are prefaced by an "H" and numbered 1-5 below (e.g. "H.1"), while their related criteria are each uniquely identified by the hypothesis number, followed by the criteria number (e.g. "1.1" for Hypothesis 1, Criteria 1). These codes are used in the evaluation in Section 4.2.5.

H.1: The TaaS tool reduces the need for tester education and training.

1.1 The tool is packaged with extensive documentation and/or training material.

1.2 The provider is available for support.

1.3 The tool can generate test cases for the system under test.

1.4 The tool has sources that aid with interface use.

1.5 The tool eases the transition to new technologies (i.e., new operating environments).

H.2: The TaaS tool satisfies the need for tools or need for better tools.

2.1 The tool is platform independent.

2.2 The tool is operating system independent.

2.3 The tool has customizable reports.

2.4 The tool notifies users of failures.

2.5 The tool has logging and/or debugging capabilities.

2.6 The tool allows data input from external sources.

2.7 The tool is compatible with continuous integration tools.

2.8 The tool is compatible with version control systems.

2.9 The tool is compatible with build drivers.

2.10 The tool is compatible with bug tracking systems.

2.11 The tool has a framework that supports extensibility.

H.3: The TaaS tool corrects the issue of lacking, insufficient, or non-existent testing.

3.1 The tool supports automated testing.

3.2 The tool supports multiple types of testing.

3.3 The tool offers unit testing.

3.4 The tool offers regression testing.

3.5 The tool offers performance testing.

3.6 The tool offers security testing.

3.7 The tool reduces the cost of in-house testing.

H.4: The TaaS tool assists in generating realistic schedules with adequate time for testing.

4.1 The tool efficiently and effectively runs test cases using the cloud's resources.

4.2 The tool supports agile software testing.

4.3 The tool can run tests throughout the development cycle.

4.4 The tool supports continuous integration.

4.5 The tests provide feedback.

4.6 Test environments can be setup and prepared for test execution in an expedient manner.

H.5: The TaaS tool helps with communication both within the test team and between the test team and other groups.

5.1 The tool has customizable reports that can be consumed by the management team.

5.2 The tool allows developers to identify the source of issues or defects.

5.3 The tool provides a way of sharing results and data between members of the test team.

Each system is walked through and discussed in the next few sections; afterward, the tools are all evaluated against the described criteria.

4.2.3 SYSTEM UNDER TEST

The system under test was developed using Google App Engine (or GAE) [41]. GAE is a PaaS platform that permits the development and hosting of web applications written in languages such as Java, Python, and PHP. Websites and their associated databases, accessed with MySQL, datastores, accessed with NoSQL, or object storage are hosted on a Google-managed cloud. Free but restricted accounts are offered to anyone with a Google account, but fees can be paid for additional resources, including storage and bandwidth.

The GAE Admin Console allows administrators to control various aspects of the application: modifying basic configuration, adjusting performance options, viewing configured services, viewing and administering the datastore, splitting traffic between versions of the application, viewing instances, and monitoring resource utilization and statistics [42]. GAE's resource utilization charts and graphs were beneficial for comparison with performance testing tools.

Google allows a variety of plugins, permitting developers to work with familiar tools, including Eclipse, Jenkins, Maven, Git, IntelliJ, and more. The GAE's Software Development Kit (SDK) must be used to develop locally, but is freely available on the website. GAE is scalable up to seven billion requests per day.

For the purpose of this case study, a simple "To Do" application was created in Eclipse Kepler using Java 1.7, based on the tutorial written by Lars Vogel [43]. The front end was a simple JavaServer Page (JSP), written in Java and HTML, with a Cascading Style Sheet (CSS) used to modify the look of the page. The application consists of four servlets to allow for adding and deleting of "To Do" items and for logging in and out. GAE has a built-in method for logging in and out using Google accounts, but an oversimplified custom method was used to have better control when testing. The Java Persistence Application Programming Interface (API) was used to help maintain data in the session using a schemaless object datastore, which is queried using the Java Persistence Query Language (JPQL). The JPQL queries' syntax resembles that of Structured Query Language (SQL) queries. Code examples for the system under test can be found in Appendix B.

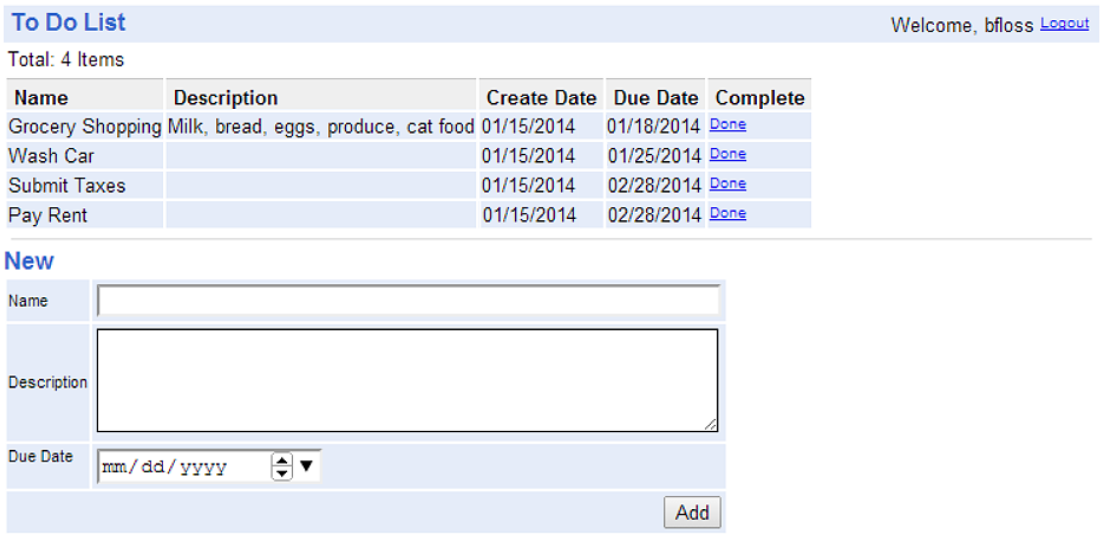


Figure 4.3: To Do application.

4.2.4 ANALYSIS OF TOOLS

The following sections describe experiences with using each tool, detailing the learning process, test creation and execution, and results. As previously discussed, these tools have various methods of use and access; however, not every aspect of the tools is explored for the purpose of the case study. Due to the limited scope of the case study and the small scale of the system under test, an additional subsection references enterprise-level success stories with each tool.

Sauce Labs

Getting started: From the Sauce Labs homepage, new users can quickly sign up for an account. Once logged in, a dashboard displays a table that will contain the executing/executed tests and information about the account itself, including account type, number of parallelized tests permitted, and number of minutes remaining. In the left module, the access key is also provided, which is used for connecting the externally generated tests to the user's account. A screenshot of the dashboard is shown in Figure 4.4. A closer look at the dashboard is provided later in this chapter.

Sauce Labs offers two general methods of creating tests. The first is to write the tests from scratch and run them through a build driver, such as Maven. The other is to record tests using Selenium Builder [44]. With no prior knowledge of writing tests, Selenium Builder is the easiest option. Simply install the Firefox plugin, click on the logo in the bottom right corner of the browser, and start recording. Instructions are available on both the Sauce Labs and Selenium Builder's web-

sites. Writing tests from scratch requires more knowledge and can become complicated depending on experience.

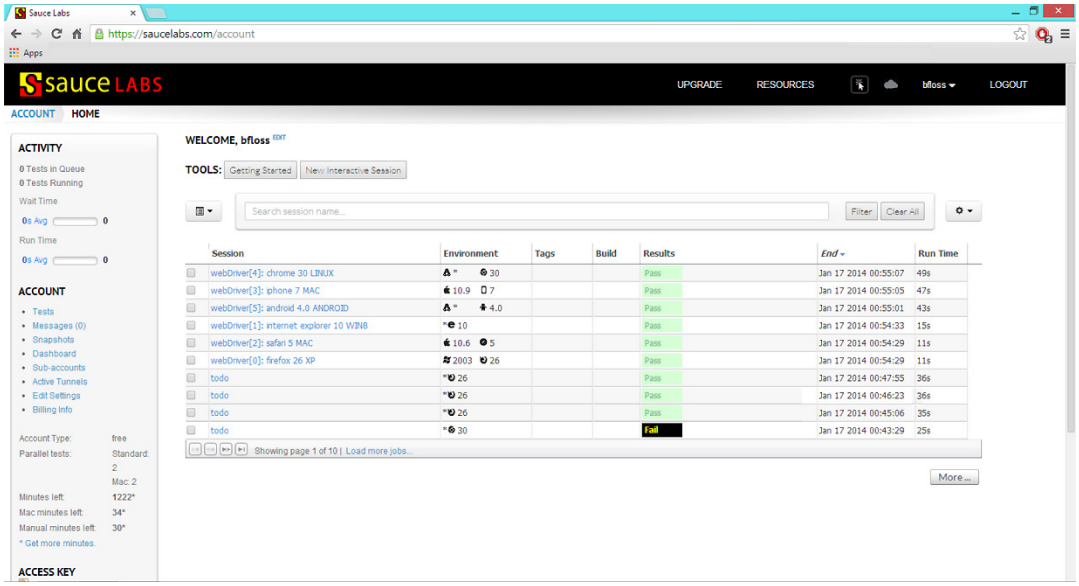


Figure 4.4: Sauce Labs dashboard.

Fortunately, Sauce Labs provides extensive documentation to help. Tutorials exist for Selenium 1 and 2 using Java, Python, Ruby, and more languages; JS Unit; and various continuous integration tools. The tutorials and the examples they provide are fairly simplistic in their testing capabilities, though, and additional research on unit testing and the scripting language of choice is likely required for those with no experience.

Writing tests from scratch: Having experience with JUnit (albeit no experience with Selenium or Maven), Java was selected as the language for writing unit tests. Using the tutorials provided by Sauce Labs and some additional reading into Maven and Selenium 2 (WebDriver), two tests were written. The first performs a test on the basic functionality of the system under test in a single environment (Windows 8 with Firefox version 26), while the second performs a simple test that verifies the title of the system under test under six separate environments to explore parallelization:

- Windows XP with Firefox version 26
- Windows 8 with Internet Explorer version 10
- Mac OS X 10.6 with Safari version 5

- Linux with Chrome version 30
- iPhone iOS 10.9 with Safari version 7 using portrait orientation
- Android tablet with Android browser version 4.0 using portrait orientation

A simple example of a Selenium 2 test case written using Java is shown below. This example is offered by Sauce Labs, but has been commented to help with understanding the functions. Due to their length, the code for the tests written for this case study is available in Appendix C. The Maven information for running these tests is also available there.

```
public class WebDriverTest {

  private WebDriver driver;

  /**
   * The setup function sets the environment and creates the
   * RemoteWebDriver that connects to the user account by using
   * the account's username and access key. The access key was
   * changed to <access_key> in this example for security, but is
   * available on the Sauce Labs dashboard.
   */
  @Before
  public void setUp() throws Exception {
    DesiredCapabilities capabilities = DesiredCapabilities.firefox();
    capabilities.setCapability("version", "17");
    capabilities.setCapability("platform", Platform.XP);
    this.driver = new RemoteWebDriver(
        new
URL("http://bfloss:<access_key>@ondemand.saucelabs.com:80/wd/hub"),
        capabilities);
  }

  /**
   * The actual testing is performed here. In this case, the
   * test simply goes to Amazon and verifies the title.
   */
  @Test
  public void webDriver() throws Exception {
    driver.get("http://www.amazon.com/");
    assertEquals("Amazon.com: Online Shopping for Electronics, Apparel,
Computers, Books, DVDs & more", driver.getTitle());
  }

  /**
   * The driver is closed here.
   */
  @After
  public void tearDown() throws Exception {
    driver.quit();
  }
}
```

Writing two tests from scratch while learning Selenium, Maven, and Sauce Labs and working part-time took approximately one week to complete with some trial and error. While not an overly long period of time for an experienced developer, the time required to complete a single test, especially for a more complex system or a tester with less development experience, could grow significantly. For these scenarios, writing a test script using Selenium Builder offers significant advantages.

Working with Selenium Builder: Selenium Builder is an open source tool for writing Selenium 1 or 2 test scripts by recording actions on a Firefox browser. The Firefox plugin can be installed from the Sauce Labs website. Once installed, the tool is available from the green LEGO®-like brick at the bottom right of the browser, as seen in Figure 4.5. Clicking on the brick brings up the Selenium Builder tool in a separate window, shown to the right in the figure.

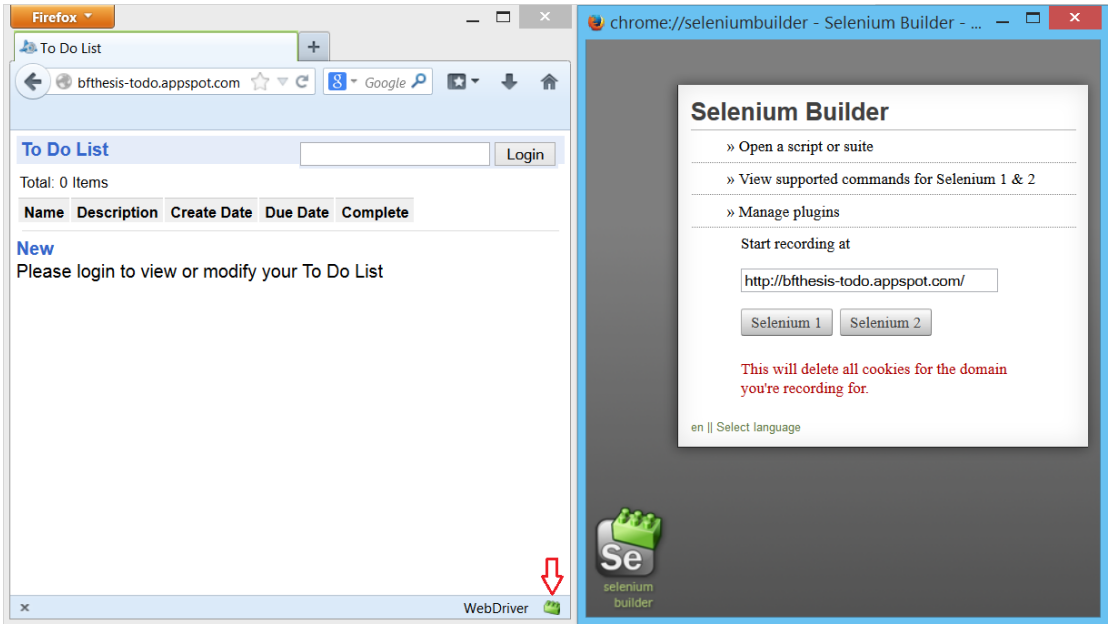


Figure 4.5: Selenium Builder tool.

To start recording, simply select the button showing the version of Selenium desired and then perform actions on the browser showing the system under test. At any point, verifications can be recorded by selecting the "Record a verification" button and then clicking on the field that needs to be verified. Various steps can be recorded manually in the tool, including navigation, input, assertions, verifications, waits, stores, and even miscellaneous items including printing, screenshots, and working with cookies. Figure 4.6 shows a recording in progress.

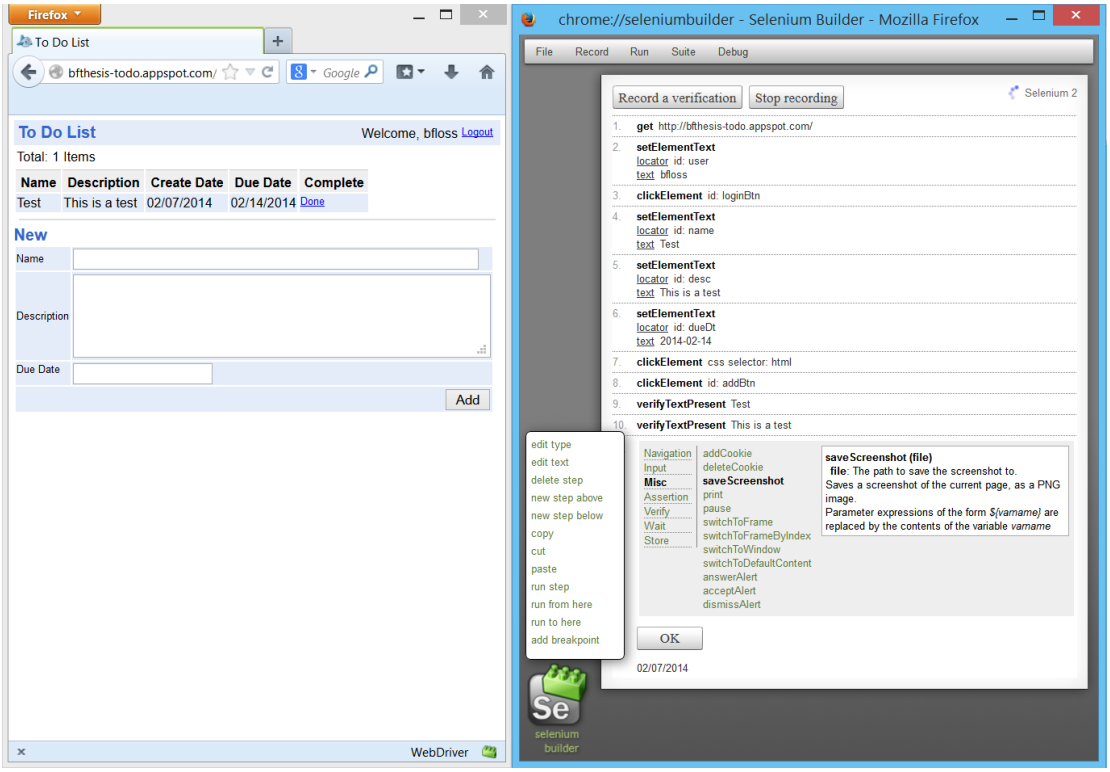


Figure 4.6: Selenium Builder recording in progress.

Once the test is completed, simply press the "Stop recording" button. From here, tests can be saved individually or added to a suite, run locally or on Sauce OnDemand, or debugged. Figure 4.7 shows the prompt for running parallel tests in Sauce Labs from Selenium Builder. The following section discusses the Sauce Labs dashboard. The information in that section applies to both the manually written and recorded tests.

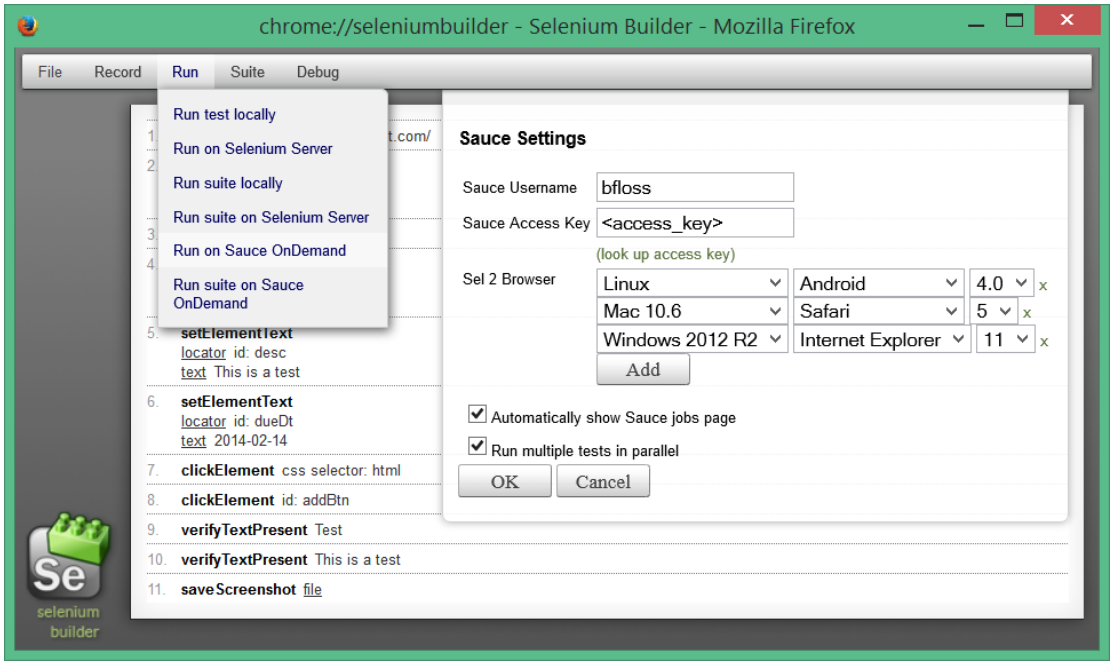


Figure 4.7: Running tests from Selenium Builder.

Viewing the results: The Sauce Labs dashboard shows all tests that have been, or are in the process of being, executed. Figure 4.8 shows a closer look at the lists of tests displayed in Figure 4.4. For each test, the session, the environment information, the results, and timing information are displayed. In the example, the session name and results were set in the code (shown and commented in Appendix C), where the pass or fail is determined by the assertions—if all assertions pass, the results declare the test to be passed.

Session	Environment	Tags	Build	Results	End	Run Time
webDriver[4]: chrome 30 LINUX	* 30			Pass	Jan 17 2014 00:55:07	49s
webDriver[3]: iphone 7 MAC	10.9 7			Pass	Jan 17 2014 00:55:05	47s
webDriver[5]: android 4.0 ANDROID	* 4.0			Pass	Jan 17 2014 00:55:01	43s
webDriver[1]: internet explorer 10 WIN8	* 10			Pass	Jan 17 2014 00:54:33	15s
webDriver[2]: safari 5 MAC	10.6 5			Pass	Jan 17 2014 00:54:29	11s
webDriver[0]: firefox 26 XP	2003 26			Pass	Jan 17 2014 00:54:29	11s
todo	* 26			Pass	Jan 17 2014 00:47:55	36s

Figure 4.8: Sauce Labs test table.

Clicking on a session link will bring up a page dedicated to that test, as shown in Figure 4.9. The different tabs are explained below:

- Commands: The Commands tab shows every command that was called during the script, including information about the input or verifications. At each step, a screenshot was automatically taken, which is viewable on the right.
- Screencast: The Screencast shows a recording of the test, allowing users to watch the test playback from start to completion.
- Selenium log: The Selenium log is available for the test execution.
- Metadata: The Metadata tab shows all metadata associated with the test. Additionally, various logs and media are available for download. In this case, a Selenium, Sauce, and Firefox log are available, along with one video and 13 screenshots.

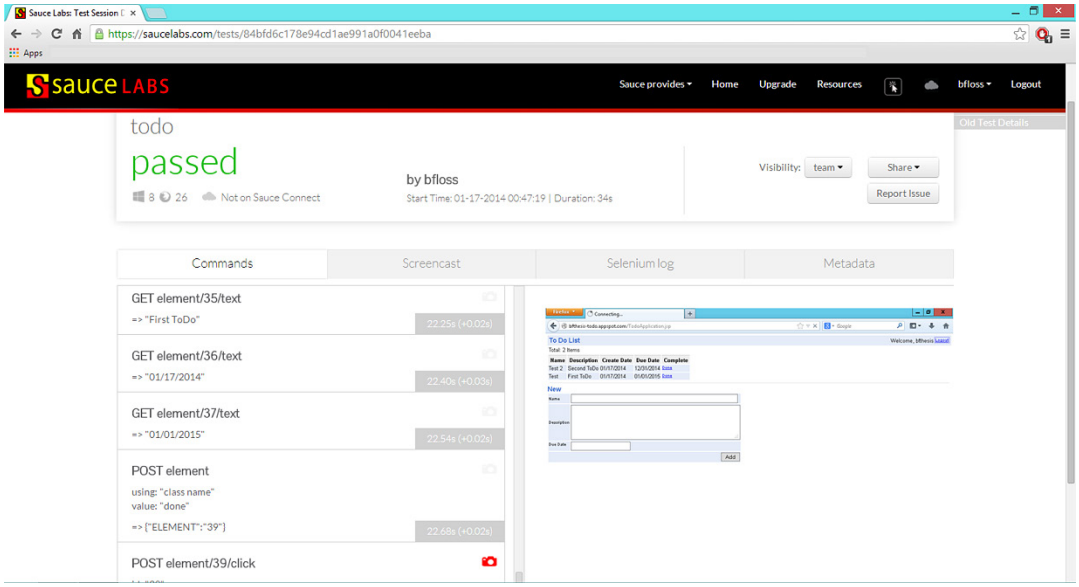


Figure 4.9: Sauce Labs test results.

Sauce at the Enterprise Level: Sauce Labs has been successfully integrated at the enterprise level. Several case studies are available on their website, but the benefits for a select few are listed below:

- Mozilla [45]: With 18 projects and 80 suites, Mozilla can more reliably and quickly run thousands of tests each day with less stress. Test suites that took 8 minutes to run locally take only 2 minutes to run on Sauce due to parallelization. With this in mind, Mozilla expanded their testing to include more browsers and more operating systems.

- Okta [46]: Okta is a leader in enterprise identity management, serving 300,000 people through its cloud-based system. Okta was previously using local Selenium tests, but migrated to Sauce Labs and saw immediate benefits. Key improvements include a massive reduction in time taken to run the core test suites (from 24 hours to 10 minutes), a massive reduction in time taken to debug tests (from 3 days to 3 hours), and a massively expanded scope thanks to running hundreds of tests in parallel.

- Eventbrite [47]: Eventbrite immediately noticed improvements by switching to Sauce Labs in terms of development, testing, and debugging time. They connected to Jenkins to provide continuous integration support and run a smoke test job containing 20 tests as a sanity check with every build. When successful, the smoke test is followed by a suite of 700 Selenium tests using 35 concurrent threads focusing on Chrome, Firefox, and Internet Explorer. They claim that stability has never been better.

Other corporations using Sauce Labs include Yelp, Dropbox, BBC, Adobe, and Travelocity.

SOASTA CloudTest Lite

Getting started: On the SOASTA homepage, new users can sign up for a free account. Once the required information is submitted, an email with instructions is sent to the user. SOASTA provides a CloudTest Lite virtual machine image and states that the minimum hardware requirements are 4 GB RAM (with virtual machines requiring 2 GB), a 64-bit processor, and 20 GB of free disk space. The image can be installed in VMware Player on Windows and Linux or VMware Fusion or Parallels on Mac OS X. Alternatively, the image can be deployed directly onto a VMware ESX, ESXi, or vSphere environment using the Open Virtualization Archive (OVA) package provided. SOASTA's 64-bit virtual machine is built on the CentOS ("Community Enterprise Operating System") Linux distribution, but no Linux experience is necessary to use CloudTest Lite.

For this case study, VMware Player was installed on a Windows 8 machine that exceeded the minimum software and hardware requirements. The VMX file (the primary configuration file for a virtual machine) was opened in the Player (Player > File > Open…) to start CloudTest Lite. For Intel machines running a 64-bit virtual machine, Intel Virtualization Technology must be enabled in the BIOS; otherwise, VMware Player may throw an error or will simply display a black screen and close.

Once the virtual machine is running, the user must enter a license activation key. At this point, the screen shown in Figure 4.10 should be displayed. Navigate to the given web address on the local machine's browser to access the CloudTest Lite dashboard. The required username and password is also given in the instructional email. Once logged in, the dashboard shown in Figure 4.11 should be displayed.

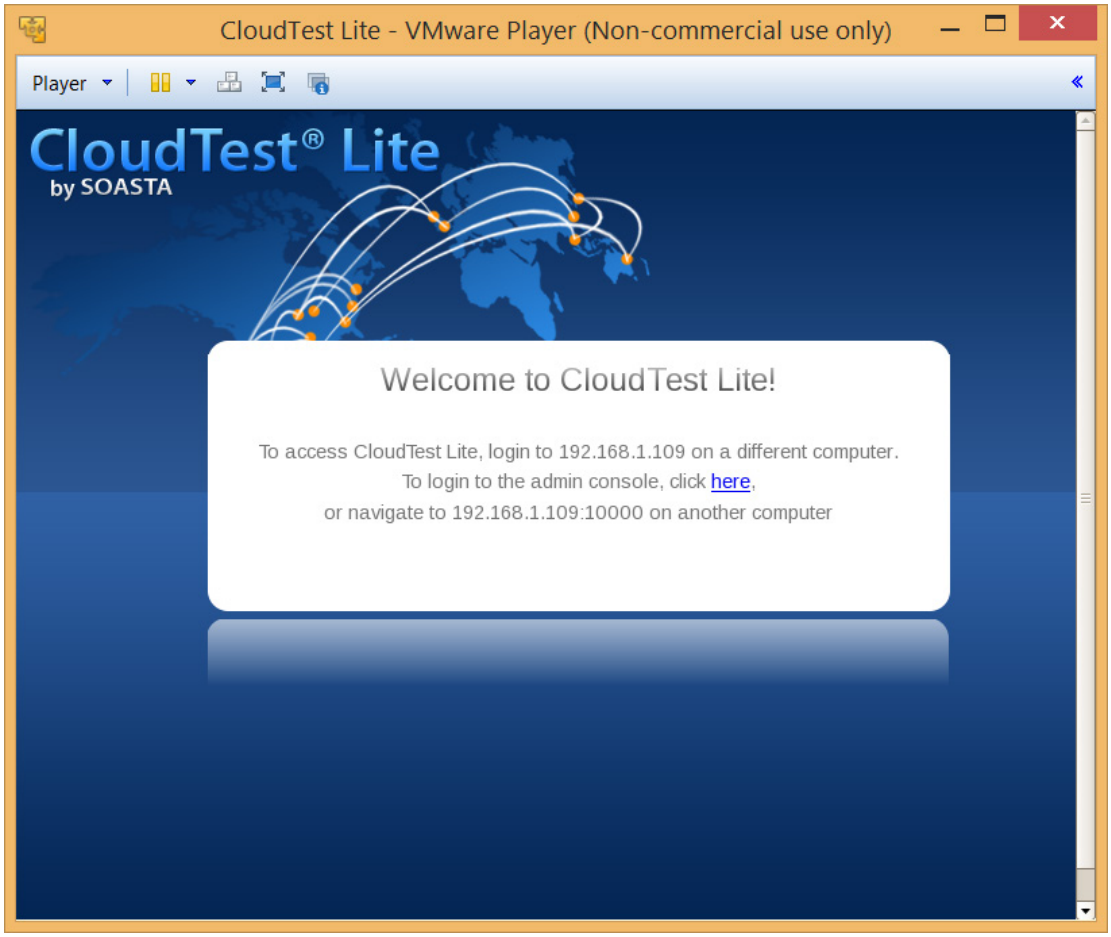


Figure 4.10: CloudTest Lite in VMware Player.

SOASTA CloudTest Lite's dashboard contains a significant amount of information, which may be overwhelming to first-time viewers. The bottom half of the center module offers an abundance of information, including forums, a knowledge base, training videos, documentation, and support. SOASTA's system is not the most intuitive, but they do provide an incredible amount of resources to help new and experienced users. Additionally, users should occasionally receive emails about seminars, including "Getting Started" webinars.

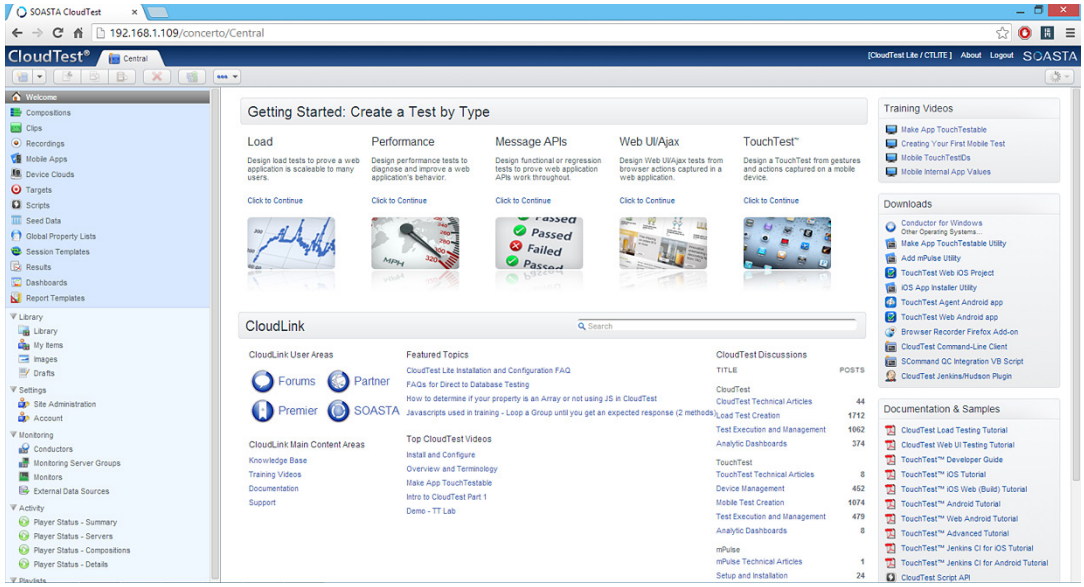


Figure 4.11: CloudTest Lite dashboard.

While the basic steps to creating and running a test are described below, this case study was performed after watching several training videos on recording and playing test scripts using CloudTest Lite. These videos were extremely helpful, walking users through the tool and clearly explaining the actions being performed. They are highly recommended for learning CloudTest Lite.

Creating tests: To enable recording, users must first install the SOASTA Conductor, which is available in the right module of the dashboard under "Downloads." Once installed, the Conductor should automatically start and appear in the Window's taskbar, located near the date and time, with the following icon: . If the Conductor did not start automatically, right click on the icon and click "Start."

Once the Conductor is running, select the type of test to create, using the links provided in the center module. In Figure 4.12, the steps are shown for creating a performance or load test. First, a recording must be created and saved. A completed recording can be converted into a sequenced test clip. Saved test clips can be combined to form a test composition, which are executed to perform testing.

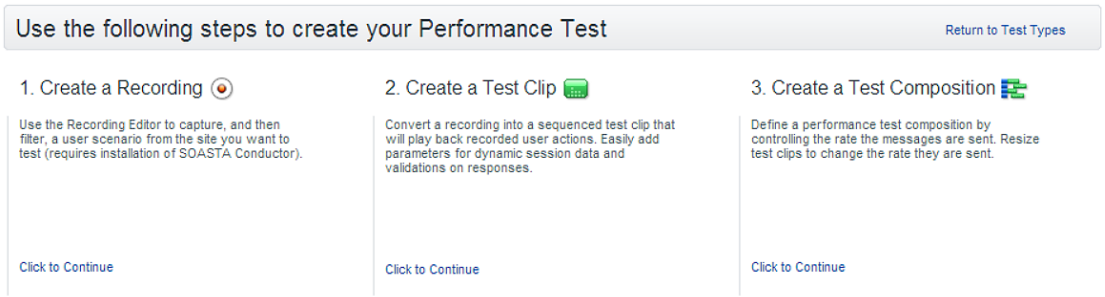


Figure 4.12: CloudTest Lite Performance Test steps.

To begin, click the recording option. In the Target Definition Wizard, select the type: HTTP, SOAP with a WSDL URL, WebUI/Ajax, Native/Mobile App, Existing Target(s). In this case study, an HTTP recording was used. On the next screen of the wizard, fill out the target name, the location (URL), and any required authentication information to continue. Upon completion, the Test Clip screen should be displayed with an option to record new scripts. At this point, SOASTA will verify that a Conductor is running on the local machine.

Once the recording begins, the Conductor will record all actions that send data across the network. For this reason, SOASTA recommends closing any programs that access the internet. All actions performed against the system under test should be done using a separate browser, such as Firefox or Chrome, that has first been cleared of all history and data and set to the "about:blank" page prior to recording. As actions are recorded, they appear as icons across the page. Each icon can be clicked on to view the request and response headers and bodies. The response body can be viewed as an HTML page, although no style sheet is applied. A completed recording is shown in Figure 4.13.

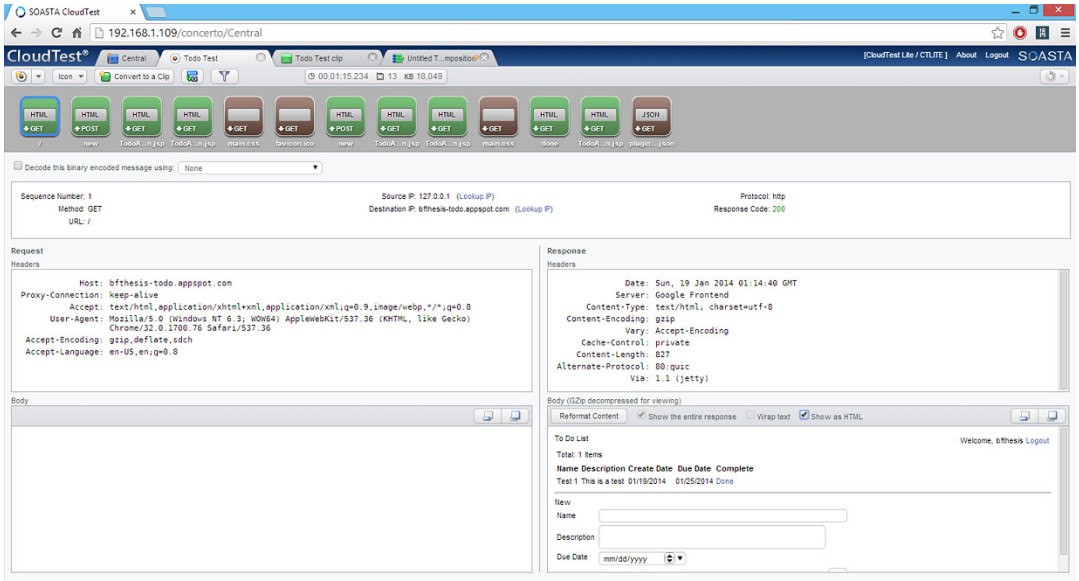


Figure 4.13: CloudTest Lite recording.

Once the recording is complete, the steps can be converted to a test clip. As seen in Figure 4.14, the test clip screen displays the individual steps that were performed. At this point, the script can be modified. One option is to add validations to the script, allowing the verification of data and causing script failures under certain conditions. Another is to generate or import test data. When the test script is prepared for use, it should be saved.

In the test composition screen, saved tests can be dragged to various tracks. The track is sequenced, causing tests to run one after another. The various tracks can run concurrently. By clicking on the icon in each track, the number of users can be set. Clicking on the icon within each clip allows the number of repetitions to be set. Ramp up times and pacing can be set within the properties for each clip. A test composition is shown in Figure 4.15. Three instances of the same test clip are used, two running in parallel and one running sequentially, each with varying numbers of users and repetitions. Once the composition is complete, the tester can press the play button and execute the script.

The process for creating and executing the first test took longer in SOASTA than the other tools examined. A few hours were required due to the complexity of the tool and the need to watch the educational material.

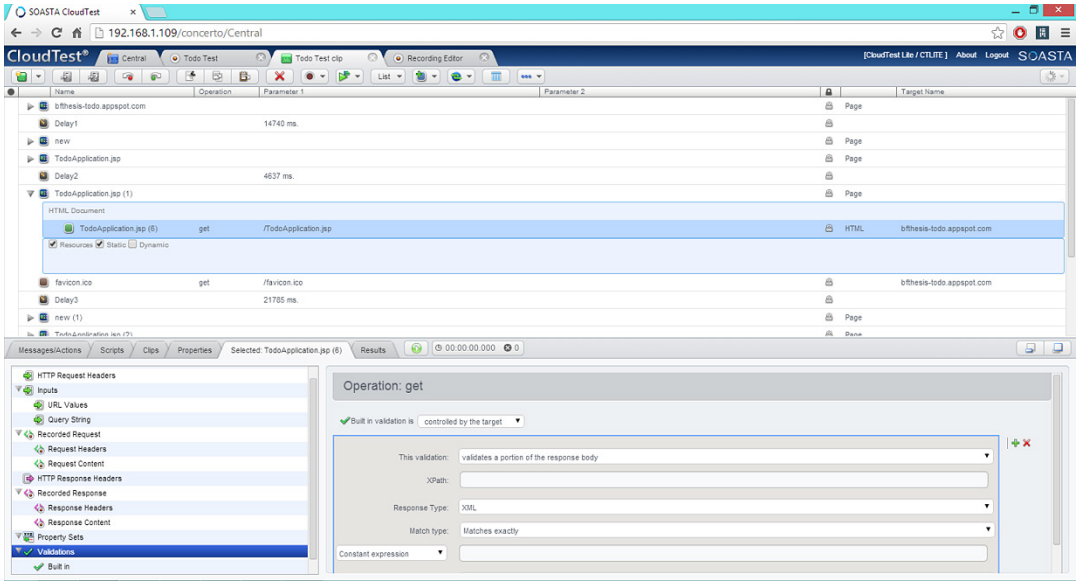


Figure 4.14: CloudTest Lite test clip.

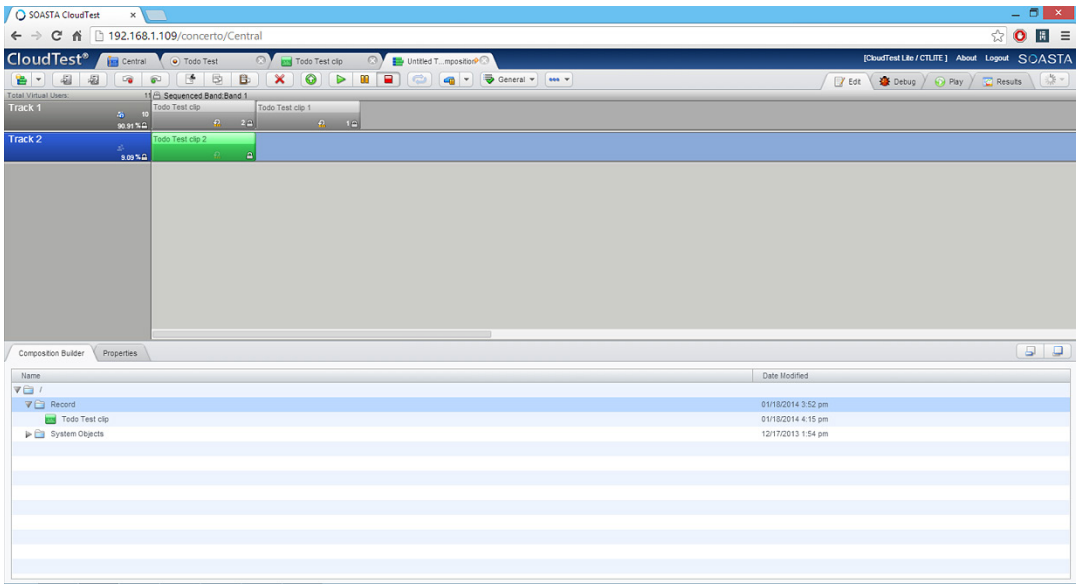


Figure 4.15: CloudTest Lite test composition.

Running the test and viewing results: During test execution, various dashboards are available. The Results tab, shown in Figure 4.16, displays the steps as they are called and offers basic information for each action, including duration, average response time, bytes sent and received, and

throughput. This page can be modified to include a large variety of widgets, as seen in the left module. Additionally, preconfigured or custom dashboards can be added in new tabs, such as SOASTA's Load Test Summary shown in Figure 4.17.

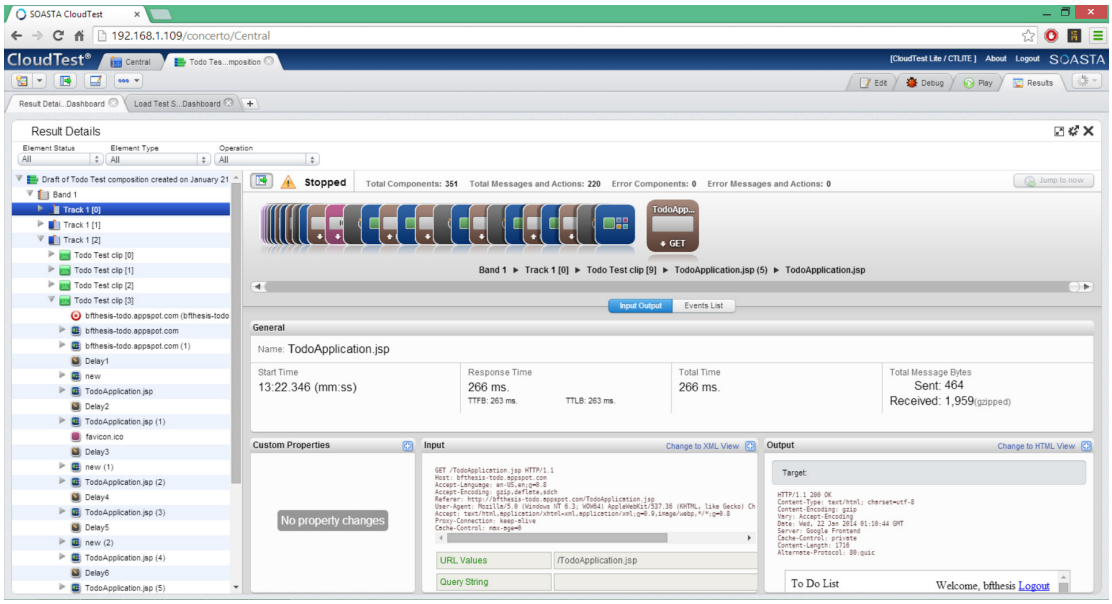


Figure 4.16: CloudTest Lite test results.

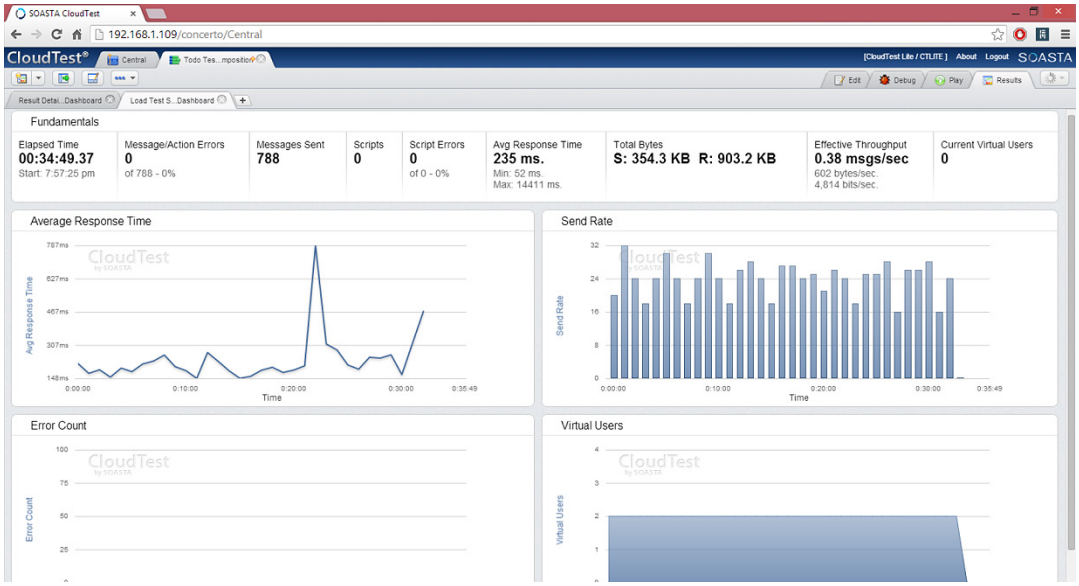


Figure 4.17: CloudTest Lite load test summary.

Google App Engine has its own charts showing usage over time. Figure 4.18 shows a summary of requests on the To Do application during CloudTest Lite's test execution. As expected, this graph of total requests over time closely follows SOASTA's Send Rate graph shown in the top right of Figure 4.17.

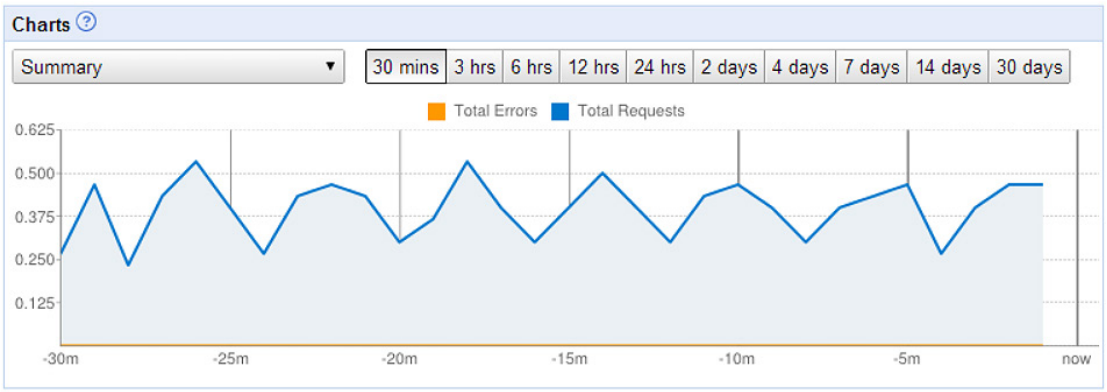


Figure 4.18: GAE Request Graph—CloudTest Lite.

SOASTA at the Enterprise Level: SOASTA CloudTest is built for enterprise testing and has been successfully used to test massive undertakings including the following:

- London 2012 Olympics [48]: Six months prior to the 2012 Summer Olympics, the website team began working with SOASTA in hopes of finding all possible bottlenecks to prevent catastrophic situations that would impact the user experience and the London 2012 brand. SOASTA performed over 500 tests on the site and mobile apps, simulating over 400,000 concurrent users from across the globe. The team stated that they would have required hundreds to thousands of servers and weeks of setup to perform a single test, but they instead simulated 100,000 users within minutes. By the end of the Games, the following totals were reached:
  - 1.3 PB of data served
  - 1.7 billion object requests
  - 46.1 billion page views (HTML/HTMX)
  - At its peak, 104,792 page views per second on the web and 17,190 page views per second on the mobile application
- Microsoft [49]: SOASTA assisted Microsoft in testing its Windows Azure Platform in 2010. The customer site, Office.com, was tested using 10,000 concurrent virtual

users with domestic network latency. The test planning took only three days, while the tests themselves took only three hours. SOASTA managed the testing process while Microsoft employees used SOASTA's OLAP engine to observe the real-time results. Office.com engineers agree that they spent far less time testing and were able to count on higher reliability thanks to the scaling capabilities of CloudTest.

- The Kentucky Derby [50]: The United States Thoroughbred industry faces increased technological challenges online due to internet wagering, internet betting, viewing, and participation. Due to major spikes in traffic, Churchill Downs, Inc. decided to test website performance prior to Derby Day to better understand the performance and scalability of their own infrastructure, seek improvements, and achieve 10–12,000 HTTP hits per second with optimal response times. Their existing performance testing practice used open source tools with only two web servers. They successfully sought help from CloudTest to emulate anticipated volume from outside their firewall with little time remaining prior to race.

Other SOASTA users have included Activision, SAP, Lenovo, Intuit TurboTax, and more.

BlazeMeter

Getting started: Of the TaaS tools examined, BlazeMeter was the simplest to understand and use, especially when utilizing the Chrome extension to record tests. To create an account, users only need to enter their email, password, and name. Once logged in, the extension's information can be found on the "GET and POST Requests" section of the dashboard, which offers all the instructions required to use the tool. The extension can be installed from the Chrome Web Store at no cost.

Recording tests: Once installed, the BlazeMeter icon appears on the right of the address bar in Chrome. Click on the icon to setup and start the recording. The tool, shown in Figure 4.19, allows users to set a name, the concurrency, the load origin (Ireland, Virginia, Northern California, Oregon, Singapore, Sydney, Tokyo, or San Paulo), and other options. Once the record button is pressed, all actions within the browser are saved. When no further actions are required, press the stop button to end the recording.

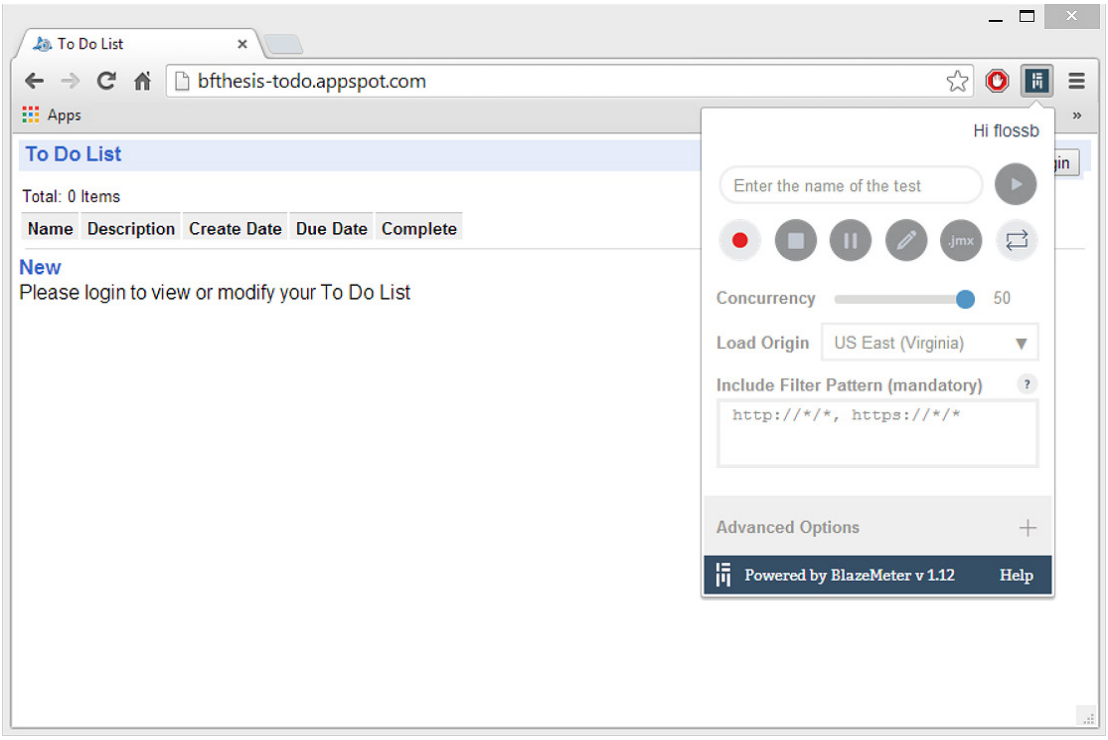


Figure 4.19: BlazeMeter Chrome extension.

If the recording is satisfactory, press the ".jmx" button to convert and save the script as an Apache JMeter file. If the recording requires editing, click the pencil button instead to view the Editor, which is shown in Figure 4.20. The Editor allows scripts to be exported as a JMX or JSON file. JMeter files can be uploaded into BlazeMeter. Using the BlazeMeter plugin, a simple test was written and prepared for execution within minutes.

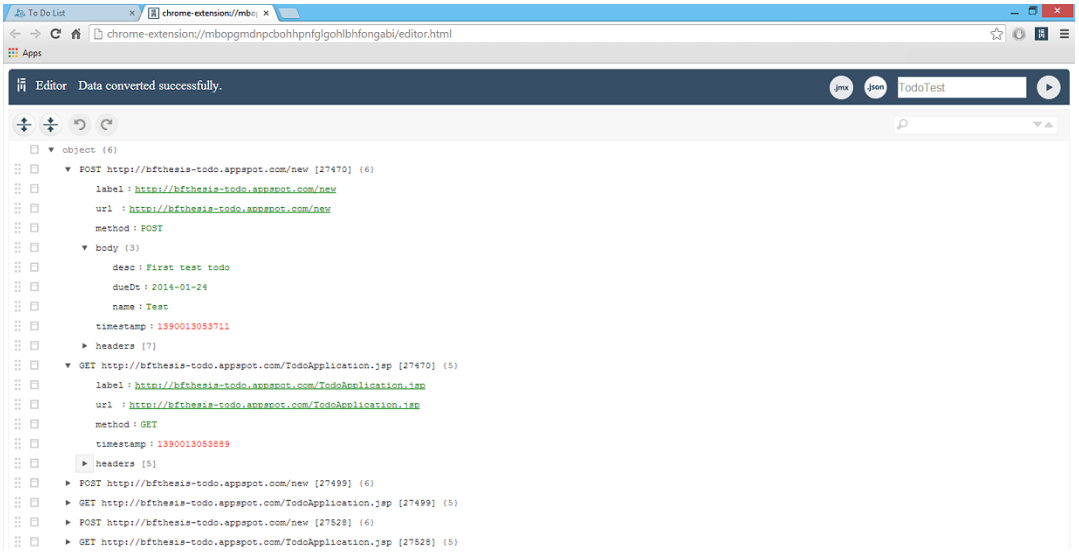


Figure 4.20: BlazeMeter script editor.

Running tests and viewing results: Using the BlazeMeter dashboard, test scripts can be uploaded in the Apache JMeter section, shown in Figure 4.21. Once uploaded, changes can be made to the number of concurrent users, the ramp up period per thread group, the number of iterations per thread group, and the duration per available thread group. When the test configuration has been set, the information can be saved as a test, which becomes available in the "Tests & Reports" page. Pressing play can run the test, shown in Figure 4.22. From the reports screen, real-time test results are displayed. Upon completion, an email is sent to the account owner to notify them that the test has completed its execution.

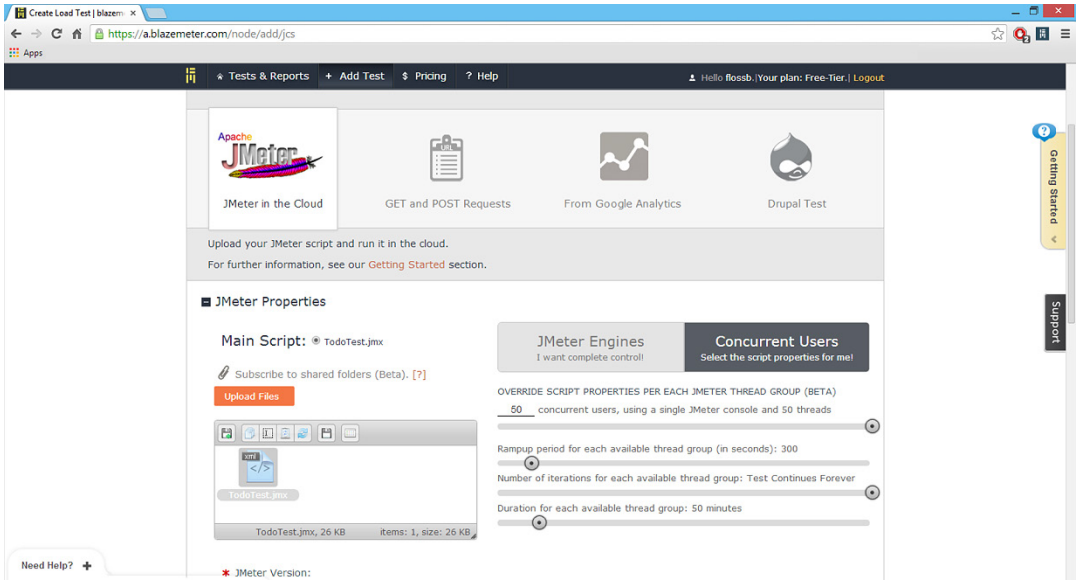


Figure 4.21: BlazeMeter JMeter test configuration.

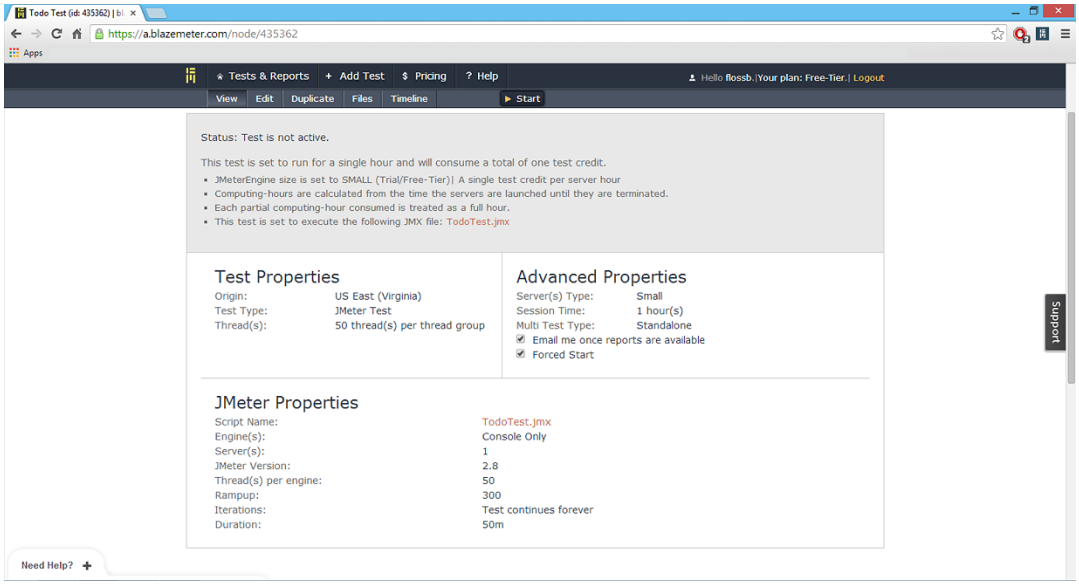


Figure 4.22: BlazeMeter test.

Reports are available on the "Test & Reports" page. Various graphs are available, comparing aspects of the test results. Results can also be compared against the results of previous tests. Examples of load results and performance monitoring graphs are shown in Figures 4.23 and 4.24,

respectively. In Figure 4.23, the thicker lines represent the current test, while the thinner lines are the previous test results. Errors and JMeter logs are also available. The errors tab is shown in Figure 4.25. In this case, errors were caused by using all of the datastore read operations permitted by the free version of Google App Engine in a single day, locking down the application until the resources reset. Figure 4.26 shows the GAE dashboard for this time period. Plugins, such as New Relic, would also be displayed here if they were configured.

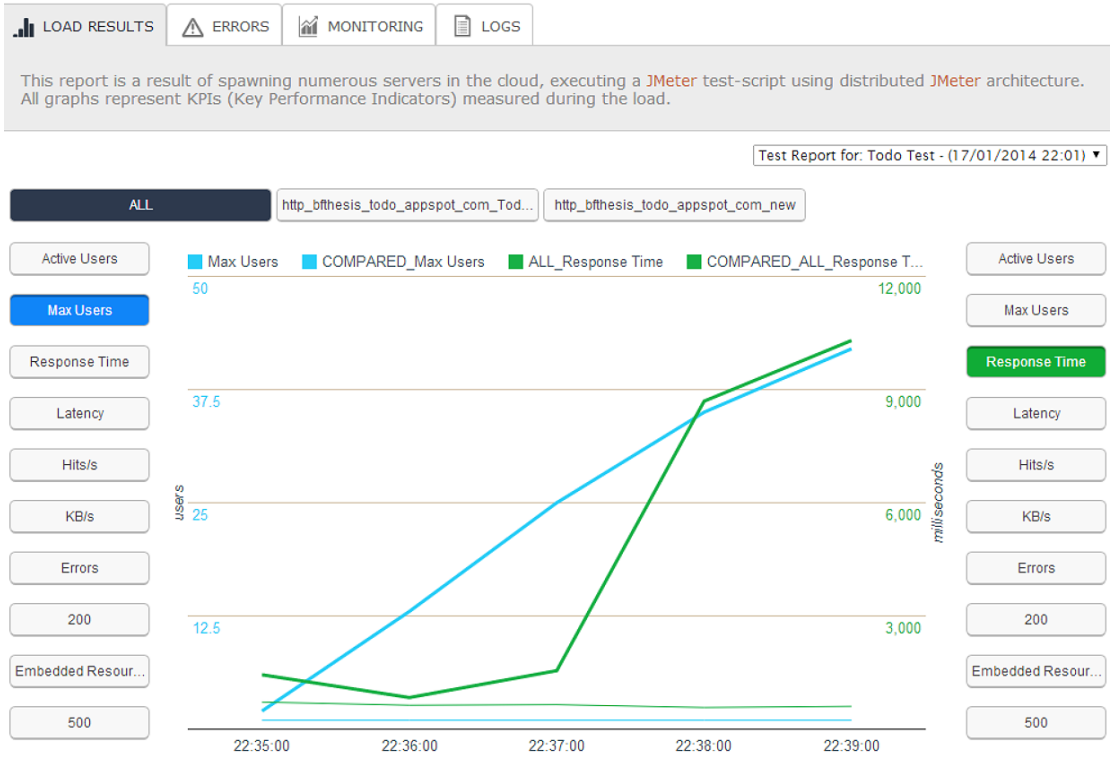


Figure 4.23: BlazeMeter reports—load results.

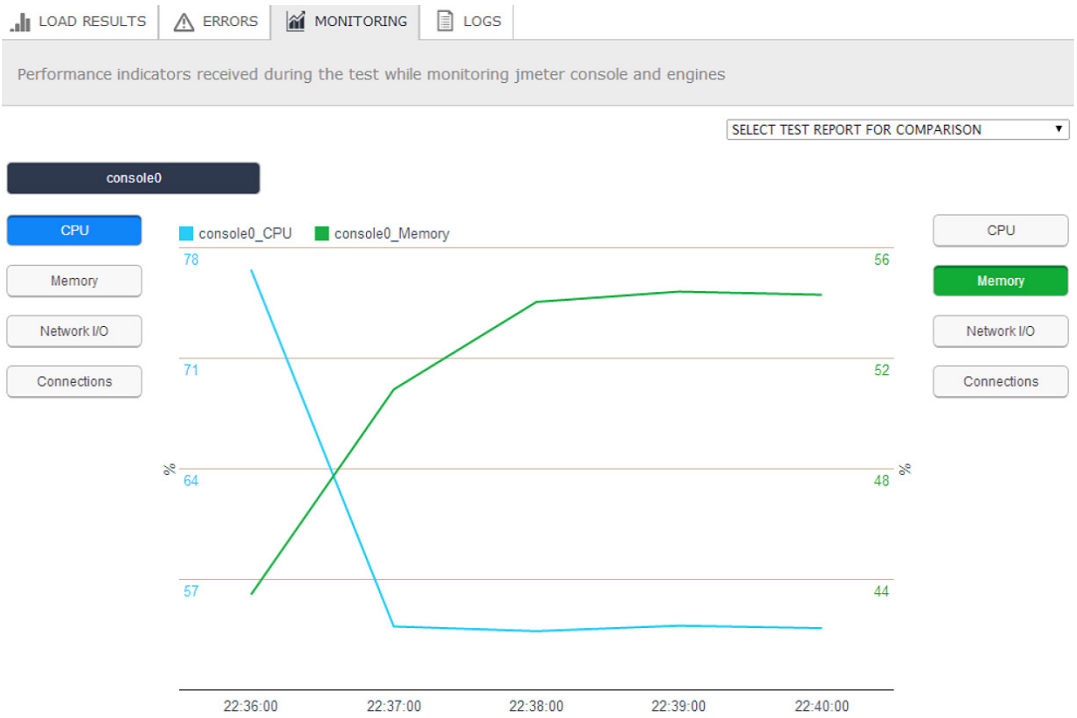


Figure 4.24: BlazeMeter reports—monitoring.

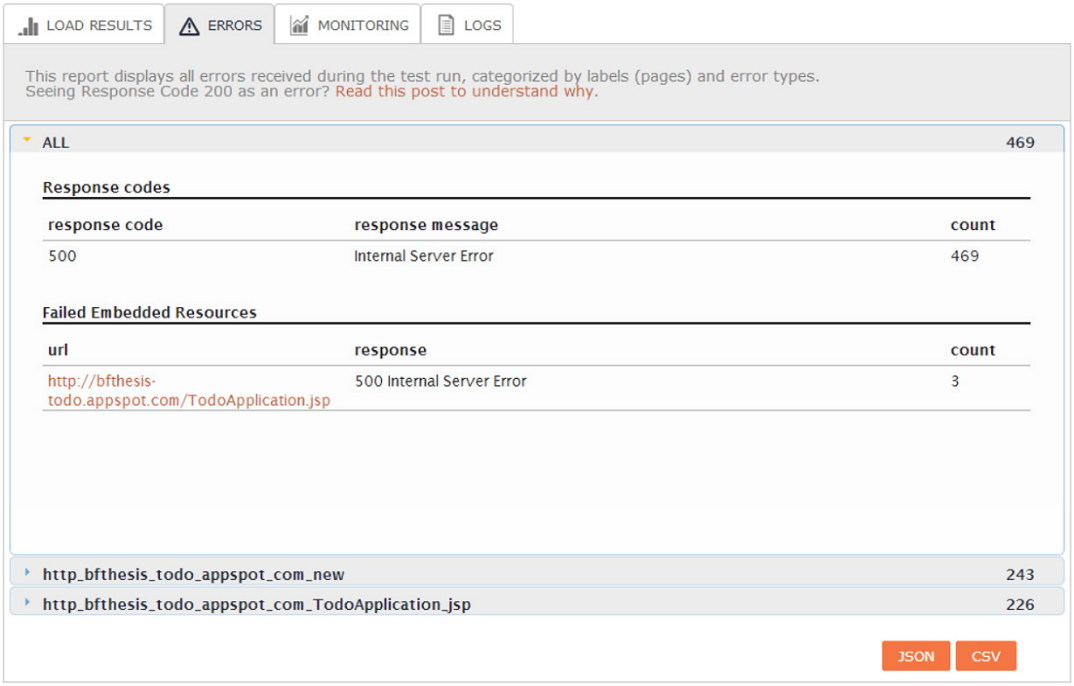


Figure 4.25: BlazeMeter reports—errors.

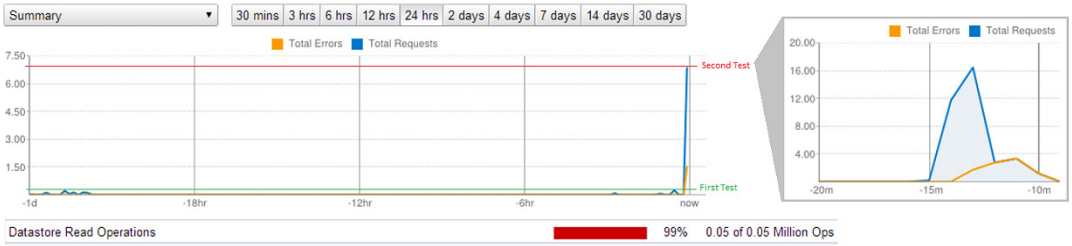


Figure 4.26: GAE Request Graph—BlazeMeter.

BlazeMeter at the Enterprise Level: Many groups use BlazeMeter at the enterprise level, including Adobe, BBC, Citibank, RE/MAX, Massachusetts Institute of Technology, and Nike. While they do not yet have case studies available on the website, they do post a selection of testimonials. Three are repeated below, all from the testimonials site [51]:

- "BlazeMeter allows us to effortlessly load test at large scale capacity. We are able to run 50,000 simultaneous user tests on a weekly basis and always head into any large-scale event with absolute confidence."—Shlomo Rothschild, Director, Divisional Information Security and Architect at InterCall

- "BlazeMeter enabled us to get test results fast and effortlessly. Waiting to provision dozens of test servers and manage the intricacies of distributing large scale load tests is too costly and simply unrealistic in traditional IT environments."—Mike Valenty, Technical Director, Double Jump Games & Maker of iWin Slots

- "After evaluating various load testing tools, BlazeMeter's solution was the obvious choice. I found it professional, scalable and most importantly—simple to use."—Shay Finkelstein, CEO, Dataphase

4.2.5 RESULTS OVERVIEW

The evaluation framework's results are displayed in Table 4.2. The criterion is derived from the hypothesis explained in Section 4.2.2. The results are primarily pass or fail, where a checkmark denotes a pass and a blank cell denotes a failure. A pass means that the criterion is satisfied for the tool, while a failure means that the criterion was not met. In some examples, an asterisk is given. The asterisk implies that the criterion is only partially met. Each partial pass is described after the table of results.

Table 4.2: Evaluation results overview

Criterion	Sauce Labs	SOASTA	BlazeMeter
H.1. The TaaS tool reduces the need for tester training and education			
1.1	✓	✓	✓
1.2	✓	✓	✓
1.3	*	*	✓
1.4	✓	✓	✓
1.5	✓	✓	✓
H.2. The TaaS tool satisfies the need for tools or need for better tools			
2.1	✓	✓	✓
2.2	✓	✓	✓
2.3	*	✓	✓
2.4	✓	✓	✓
2.5	✓	✓	✓
2.6	✓	✓	✓
2.7	✓	✓	✓
2.8	✓		
2.9	✓		

2.10			
2.11	✓	✓	✓
H.3. The TaaS tool corrects the issue of lacking, insufficient, or non-existent testing			
3.1	✓	✓	✓
3.2		✓	✓
3.3	✓		
3.4			
3.5		✓	✓
3.6			
3.7	✓	✓	✓
H.4. The TaaS tool assists in generating realistic schedules with adequate time for testing			
4.1	✓	✓	✓
4.2	✓	✓	✓
4.3	✓	✓	✓
4.4	✓	✓	✓
4.5	✓	✓	✓
4.6	✓	✓	✓
H.5. The TaaS tool helps with communication both within the test team and between the test team and other groups			
5.1	✓	✓	✓
5.2	✓	✓	✓
5.3	✓	✓	✓

Three criteria had partial passes given. Each is described below:

- **[1.3] The tool can generate test cases for the system under test:** Of the three tools, only BlazeMeter has the option for the fully automated generation of test cases (using Drupal or Google Analytics). However, all of these tools offer assisted test case generation in the form of test recording and playback.

- **[2.3] The tool has customizable reports:** By default, Sauce Labs will only state whether a test has completed its execution—not whether it passed or failed. Verifications and assertions will only be displayed in the table of results if the information is passed from the script, requiring some programming. Updating the script to contain

this information is fairly simple, though, and Sauce Labs does provide a great deal of detail with the screenshots, videos, and logs.

4.2.6 THREATS TO VALIDITY

While this case study attempted to provide a view of several TaaS tools used in realistic scenarios, several factors could challenge the results. For example, the hypothesis may be incomplete. An incomplete hypothesis may cause the results to only apply to certain scenarios, beyond those evaluated in this case study or those discussed in the case studies from enterprise-level users. Individuals and teams may have a very specific set of needs, which may or may not be met despite the general requirements being satisfied.

Furthermore, the tools selected may not meet the requirements of the prospective users. While unit and performance testing were both considered being "in demand" according to the HPST survey, the teams open to TaaS tools may instead require other forms of testing or prefer that the tools be packaged with more features.

Not all features of the tools discussed were examined and evaluated. These tools have many features, not all of which could be examined during the course of this case study due to time constraints and restrictions on the number of tests that could be run per month on the free plans. These other options were described based on their advertised features, which may or may not be entirely accurate in terms of offerings or quality.

Several of the criteria require subjective responses, particularly in the area of education and training. These determinations were made as objectively as possible, based on experiences within this case study and the enterprise-level case studies. Nevertheless, this leaves room for differing opinions or error.

4.3 GAP ANALYSIS

This section provides a gap analysis based on the needs of the industry and the capabilities of the TaaS tools. The TaaS tools are discussed together, with examples of strengths and weakness from each being cited. Additionally, examples from the academic literature are included to showcase potential improvements and solutions that are already being researched and developed.

4.3.1 UNSATISFIED HARD PROBLEMS

As shown in Table 4.2, the TaaS tools satisfied the majority of the criteria. Nevertheless, improvements can still be made in several areas.

Fully automated test case generation

While all three tools examined allowed the recording and playback of tests, only one offered the ability to generate test cases automatically. The issue of tester education and training could be greatly reduced by building tools with the ability to create tests without any assistance, even if the users need to supply test data. For areas such as functional, regression, and unit testing, this may not be feasible. However, load, performance, and security testing tools should have some automatic testing capabilities, even if requiring additional plugins to function. BlazeMeter's incorporation of Google Analytics and Drupal to create tests based on previous usage greatly simplifies load testing for users with less experience in testing or with heavily restricted schedules.

Compatibility with external tools

Allowing the TaaS tools to integrate with version control systems, build drivers, and defect tracking systems can simplify software development, testing, and configuration management, creating a smoother overall process. The ability to integrate with continuous integration tools like Jenkins, especially considering Jenkins's plugin library, is a great start. Working with version control systems could allow tests to be rerun whenever a developer checks in code, catching potential issues at the earliest point in time. Integration with build drivers could allow users of tools such as Ant and Maven the ability to verify that every build passes the scripted tests before moving builds into test or production environments. Compatibility with defect tracking systems could allow users to create bug reports directly from test results or perhaps even allow the automatic generation of bug reports during testing.

More types of testing

The ability to perform multiple types of testing within one tool would be a major draw for users. TaaS tools may be affordable, but combining several tools for the various aspects of testing would cause testing to, once again, become an expensive investment. Some of the tools already provide various types of testing, including SOASTA and BlazeMeter. However, the types of testing in greatest demand, according to the HPST survey, were unit, regression, performance, and security testing. While some companies claim to handle both performance and security testing, testing all four as a service would require, at minimum, three separate tools.

Greater extensibility

All three tools describe some form of extensibility, typically through access to Jenkins and, by extension, the Jenkins's plugin library or a selection of external tools, such as Selenium Builder for Sauce Labs or New Relic for BlazeMeter. While these options increase the capabilities of the existing

system, promoting even greater extensibility could provide users with even more features with less impact to the existing capabilities of the tool and less pressure on the vendor.

4.3.2 CAVEATS

There are several caveats related to the testing challenges identified in the HPST survey and the capabilities of current TaaS tools to satisfy them.

Training and education

While fully automated test case generation and extensive documentation and guides may assist those with limited experience, no amount of assistance can make up for a complete lack of testing knowledge. Despite tool capabilities, testers need to remain educated on various aspects of their work, including the application under test, testing techniques, weaknesses of software, risks of platforms or software, how programs fail, and more.

Need for tools/better tools

Most of the tools examined focus on testing web and mobile applications. Legacy desktop applications may not be testable using services at this point in time, although some academics, such as Candea, discussed testing services that accepted binaries and executables [17].

Additionally, the nature of web applications makes it difficult, if not impossible, to test the underlying code with full coverage. For example, programming languages like Java require code to be compiled into class files before deploying the application. Testing the individual functions is not possible through the class files. While much of this code should still be testable from the interface, unit testing may be more difficult in the traditional sense. On the other hand, Kosmatov's proposed structural unit testing service [24] could provide true unit testing with fuller coverage, if implemented.

Generating realistic schedules

TaaS promotes more efficient testing through the creation and execution of automated tests that can easily be rerun throughout the cycle. This should reduce the time required for testing and prevent testing from falling to the very end of the project. However, TaaS cannot help with generating the schedules themselves—just providing more time for testing to be performed.

Improving communication

The easy to read and understand dashboards should be consumable by all members of the team, allowing for better communication between the test team and other groups. The information should

allow developers to quickly pinpoint issues, while easily showing management relevant statistics and project trends. At the same time, the concerns of information sharing within the team still remain. Knowledge sharing between expert and novice employees may not be easier, but the learning process should be simpler overall.

4.3.3 ADDITIONAL CONCERNS

The academic literature provides several additional concerns that should be considered when examining the shortcomings of the TaaS tools. The challenges and needs were described in detail in Section 3.2.4. Several appear to be managed by the examined tools, such as maintaining availability, creating innovative solutions, and building trust. However, others, such as offering transparent pricing, are a bit lacking. For example, SOASTA has no pricing available on their website for the paid version of CloudTest—an issue that also applied to the majority of tools described but not included in the case study.

Another major concern is security—of the tests, of the system under test, the test data, and the results. The decision to work with a third party always creates security risks. In the case of TaaS, risks are further increased by utilizing the cloud, especially public clouds. TaaS companies make security a top priority with private accounts, secure services, data privacy guarantees, and clean installs of operating systems that are destroyed after test runtime. Nevertheless, concerns still exist about protecting proprietary information.

CHAPTER 5

Summary

Software testing is a critical phase of the software development lifecycle, responsible for helping assure system quality. However, testing suffers from a plethora of challenges, some timeless and some new, causing quality to become increasingly difficult to ensure. TaaS is an emerging method of testing software that leverages the vast and elastic resources of the cloud to offer accessible services to handle testing activities on a pay-per-use basis. TaaS promises to alleviate many of the issues of traditional testing, providing more efficient and effective testing with less effort.

The research reported in this book used the HPST series of events and a survey on problems in software testing to help determine the most challenging issues that are plaguing the industry today. An evaluation framework was then developed to compare these problems against the state of TaaS, to ascertain where a TaaS solution would be appropriate. The academic and commercial aspects of industry were both considered in detail. Ultimately, a case study was performed using three TaaS tools and a gap analysis was performed. The resulting gap was used to identify areas in need of additional work.

Significant academic work has been done in software testing over the years, but rarely are industry concerns truly considered. This work was an attempt to inform an applied research agenda based on real-world needs as described by leading practitioners. The evaluation of TaaS offerings as possible solutions to the hard problems identified is one step toward leveraging modern tools and techniques to solve persistent testing challenges.

5.1 SUMMARY OF RESULTS

The results of the HPST project are summarized according to two criteria: how well the original research objectives were satisfied, and what new contributions this work has made to the software testing community.

5.1.1 RESEARCH OBJECTIVES

There were five objectives for this work when the project began. It is instructive to examine how well each of these objectives was satisfied.

1. **Objective:** To identify and quantify recurring challenges in software testing as defined by industry professionals.

   A survey was conducted with industry professionals to elicit hard problems in software testing. The responses were quantified in Chapter 2, allowing the top issues to be

identified. The resulting top challenges were used to create an evaluation framework for use against TaaS tools.

2. **Objective:** To review and analyze the academic and commercial states of Testing as a Service (TaaS).

   The academic and commercial literature on TaaS was surveyed and analyzed in Chapter 3. Academic research focused on architecture and frameworks, types of testing, benefits and challenges, and more. The commercial aspect described the current industry offerings.

3. **Objective:** To develop an evaluation framework for TaaS tools based on the previously identified hard problems in software testing.

   An evaluation framework, presented in Chapter 4, was developed with five hypotheses, based on the top five challenges identified by the survey. Related concerns and common tool criteria were tied to the framework, building up the evaluation criteria for each hypothesis.

4. **Objective:** To apply the framework against a selection of TaaS tools with the goal of evaluating their ability to overcome these challenges.

   The evaluation framework was applied to three TaaS tools: Sauce Labs, SOASTA CloudTest Lite, and BlazeMeter. The lacking areas became obvious as the evaluation was filled out for each tool. Furthermore, aspects of the academic research were applied to the tools, noting additional concerns that should be considered.

5. **Objective:** To perform a gap analysis between the current state of TaaS and the needs of the industry and to analyze that gap to recommend improvements that could benefit the industry.

   While the tools supported many of the most important needs of the industry, a gap still exists between the described challenges and the current state of TaaS. Only one of the tools offered fully automated test case generation, the tools had limited compatibility with external tools and restricted extensibility, and they should offer more forms of testing, especially when recalling the need for regression and security testing. Additionally, concerns from the academic research still exist, specifically in the area of security and the need for transparent pricing.

5.1.2 RESEARCH CONTRIBUTIONS

The research contributions presented in this book can be summarized in two parts: quantifying the hard problems in software testing as defined by members of the industry, and evaluating TaaS tools against the user-defined needs.

Quantifying Hard Problems: This book quantified the top hard problems encountered in software testing, as identified by industry professionals. While testing issues are frequently cited, rarely is the industry surveyed as a whole. While the HPST survey was limited in range, respondents came from a variety of backgrounds, providing necessary diversity. The trends were analyzed and quantified, providing a better look at the challenges plaguing the industry and the severity and impact of each.

Evaluating TaaS Tools: TaaS is a relatively new aspect of software engineering that has been touted as overcoming many of testing's current shortcomings, but limited work has been done on evaluating tools in real-world settings. Academic and commercial literature was surveyed, providing a thorough background on the field. Three tools were selected for evaluation: Sauce Labs, SOASTA CloudTest Lite, and BlazeMeter. Each was examined in depth before an evaluation framework based on the HPST survey was applied to determine whether the tools met the top needs of the industry. While the tools met the majority of the criteria in the evaluation, a gap was revealed. Shortcomings were discussed to provide recommendations for improvements to TaaS solutions.

5.2 FUTURE WORK

Based on the gap analysis results, future work needs to be done on fully automated test case generation. Of the tools examined, only BlazeMeter offered fully automated load and performance testing. While offering script recording does help those with limited testing knowledge build a test suite, the ability to automatically generate test cases can simplify testing and soften the learning curve for TaaS tools. Furthermore, generated test cases should reduce testing time, improve the quality of the system under test, and ease the burden on testers with limited experience or knowledge.

Additionally, more tools should be evaluated against the framework. Many of the tools offered limited information on their features and lacked free trials or tiers, preventing them from being utilized in the case study. Examining more TaaS tools may reveal more gaps that need to be filled.

Finally, although some software testing challenges seem invariant (e.g., education and training), some of them change over time due to the introduction of new technologies and methodologies. When the existing problems are corrected or if new, higher priority issues are introduced, the framework may be rendered inadequate for tool evaluation. For this reason, the survey should be introduced periodically to reevaluate results.

5.3 CONCLUDING REMARKS

Software testing will never be entirely free of challenges. However, many of the current and, frequently, timeless issues can be mitigated or their impacts reduced by focusing research on the needs of the industry and applying new methodologies where applicable. For example, TaaS promises to improve testing by resolving many of these existing problems, while promoting dynamic solutions which are easily capable of adapting to future needs.

With industry support, TaaS can truly revolutionize the future of testing, simplifying the creation and runtime of diverse and manageable test suites with little cost and effort on the consumers. However, to maintain this path, hard problems in software testing must be reevaluated every few years, redirecting the tool requirements as necessary to develop and maintain strong products that consider overcoming practical testing challenges a priority.

APPENDIX A

Hard Problems in Software Testing Survey

Hard Problems in Software Testing Exit this survey

Contact: Dr. Scott Tilley, stilley@cs.fit.edu

As part of our software engineering research program, we are trying to identity hard problems in software testing. These problems should be sufficiently challenging to warrant focused attention for the next five years. They should also be sufficiently important to the community such that if measurable progress were made in solving these problems, we would advance the state of the practice in software testing to everyone's benefit.

Please help us help you. Fill out as much of this survey as you can. If you provide contact information, we will provide you with the results of this survey when it is completed. For more information, please contact Dr. Scott Tilley, Professor of Software Engineering at the Florida Institute of Technology, at stilley@cs.fit.edu . Thank you!

1. Tell us about yourself:

Name:

Affiliation:

Job Title:

E-mail:

Phone:

2. Describe five software testing problems you have struggled with in your job (these could be related to process, tools, education & training, and so on):

1.

2.

3.

4.

5.

3. Are you working in any of the following areas as a tester?

- [ ] Service-Oriented Architecture (SOA)
- [ ] Security
- [ ] Cloud Computing
- [ ] Mobile / Apps

Other (please specify)

4. What topics do you think the research community should focus on in the next five years to help you do your job better?

5. Are you interested in collaborating with us? If so, on what problem area? In what capacity?

6. Write any additional comments here:

Done

APPENDIX B

Google App Engine Code Examples

By using Google App Engine's Eclipse plugin, the basic structure, including the connectivity between the application and Google's cloud, is automatically generated upon the creation of a new project. The initial project generated by the plugin is a fully deployable "Hello, world" application that can be pushed to the cloud through a straightforward dialog in Eclipse. Google's "Getting Started" guide walks users through their first application. From there, Vogel's tutorial [43] was used to create the To Do application, although several alterations were made along the way.

Due to the use of tutorials and the number of files involved, only selections from the GAE code are included below. As the purpose of this book was to focus on the TaaS tools, this appendix only aims to offer a brief overview of the code, rather than an in-depth explanation. The initial overview of the system under test was given in Section 4.3.3. The basic structure of the non-Google-generated code is shown below:

- "src" - Source folder
  - "dao" – Data Access Object (DAO) folder

    5.5.1. Dao.java
    - EMFService.java
  - "model" – The model classes folder
    - Todo.java
    - User.java
  - CreateTodoServlet.java
  - LoginServlet.java
  - LogoutServlet.java
  - RemoveTodoServlet.java
- "war" - Web Archive (WAR) folder
  - "css"
    - main.css
  - TodoApplication.jsp

The "model" folder contains the classes used for storing data. Both files contain associated variables and the necessary accessor and mutator methods. Additionally, "Todo.java" contains con-

verters for the date format required as by the create and due date fields. The "dao" folder allows data to be accessed by the application. "EMFService.java" simply sets up the instance used by "Dao.java" to query the datastore. The excerpt below is from "Dao.java" and shows how the "To Do" items are added, removed, and retrieved for the user.

Excerpt from Dao.java

```
/**
   * Adds a "To Do" record
   * @param user Name of user adding record
   * @param name Name of "To Do" record
   * @param desc Description of "To Do" record
   * @param dueDt Due Date of the "To Do" record
   */
  public void addTodo(String user,
            String name,
            String desc,
            Date dueDt)
  {
    synchronized(this) {
      EntityManager em = EMFService.get().createEntityManager();
      Todo todo = new Todo(user, name, desc, dueDt);
      em.persist(todo);
      em.close();
    }
  }

  /**
   * Removes the "To Do" record
   * @param id ID of the "To Do" object
   */
  public void remove(long id) {
    EntityManager em = EMFService.get().createEntityManager();
    try {
      Todo todo = em.find(Todo.class, id);
      em.remove(todo);
    } finally {
      em.close();
    }
  }


  /**
   * Retrieves the list of "To Do" records for the user,
   * ordered by due date.
   * @param user Name of current user
   * @return List<Todo> A list of the "To Do" records for the user
   */
  public List<Todo> getTodoList(String user)
  {

// Continued on next page.
```

```
    EntityManager em = EMFService.get().createEntityManager();
    Query q = em.createQuery("SELECT t" +
                " FROM Todo t" +
                " WHERE t.user = :user" +
                " ORDER BY t.dueDt");
    q.setParameter("user", user);
    List<Todo> todoList = q.getResultList();
    return todoList;
  }
```

The servlets are used to transport data between the application and the datastore. The servlets are defined in the "web.xml" file within the "WEB-INF" folder (part of the WAR). Below is the definition of the "CreateTodo" servlet, used to create the actual "To Do" record. The next code section shows the actual "CreateTodoServlet.java" file, which calls the DAO to complete the addition.

Excerpt from web.xml

```
<servlet>
 <servlet-name>CreateTodo</servlet-name>
 <servlet-class>main.java.com.book.todo.CreateTodoServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>CreateTodo</servlet-name>
 <url-pattern>/new</url-pattern>
</servlet-mapping>
```

CreateTodoServlet.java

Note: The package and imports were removed to conserve space.

```
@SuppressWarnings("serial")
public class CreateTodoServlet extends HttpServlet {
      public void doPost(HttpServletRequest req, HttpServletResponse
resp)
                    throws IOException {

        String user = Dao.INSTANCE.getUser().getUsername();
             String name = req.getParameter("name");
             String desc = req.getParameter("desc");
             String date = req.getParameter("dueDt");
             SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
             Date dueDt = null;
             try {
               if(date != null) {
                 dueDt = sdf.parse(date);
               }
    } catch (ParseException e) {
      e.printStackTrace();
    }

// Continued on next page.
```

```
            Dao.INSTANCE.addTodo(user, name, desc, dueDt);
            resp.sendRedirect("/TodoApplication.jsp");
        }
}
```

Within the "war" folder, the only new files are the actual webpage and the associated style sheet. The JavaServer Page (JSP) allows the creation of dynamically generated web pages, using a combination of technologies; in this case, "TodoApplication.jsp" is written in HTML and Java. The following excerpts show how the "To Do" list is obtained and displayed and how new items can be added.

Excerpts from TodoApplication.jsp

The following excerpt shows how the DAO is called to get the current user and obtain the list of "To Do" items for the user.

```
<html>
  <head>
    <title>To Do List</title>
    <link rel="stylesheet" type="text/css" href="css/main.css" />
      <meta charset="UTF-8">
  </head>
  <body>
<%
  Dao dao = Dao.INSTANCE;
  User user = dao.getUser();
  List<Todo> todoList = new ArrayList<Todo>();

  if(user != null) {
    todoList = dao.getTodoList(user.getUsername());
  }
%>
```

The next excerpt shows how the table of "To Do" items is generated using the "todoList" object created in the previous excerpt.

```
<table id="todos">
  <tr>
   <th>Name</th>
   <th>Description</th>
   <th>Create Date</th>
   <th>Due Date</th>
   <th>Complete</th>
  </tr>
  <% for (Todo todo : todoList) { %>
   <tr>
    <td><%=todo.getName()%></td>
    <td><%=todo.getDesc()%></td>
    <td><%=todo.getCreDtAsString()%></td>
    <td><%=todo.getDueDtAsString()%></td>
    <td>
     <a class="done" href="/done?id=<%=todo.getId()%>">Done</a>
    </td>
   </tr>
   <%}%>
</table>
```

The final excerpt shows the form used to add new records. The form action is defined in "web.xml" (shown previously). The data in each input field is retrieved by the servlet and sent to the DAO.

```
<form action="/new" method="post" accept-charset="UTF-8">
 <table>
  <tr>
   <td><label for="name">Name</label></td>
   <td><input type="text" name="name" id="name" size="65"/></td>
  </tr>
  <tr>
   <td valign="desc"><label for="desc">Description</label></td>
   <td><textarea rows="4" cols="50" name="desc"
id="desc"></textarea></td>
  </tr>
  <tr>
   <td valign="top"><label for="dueDt">Due Date</label></td>
   <td><input type="date" name="dueDt" id="dueDt"/></td>
  </tr>
  <tr>
   <td colspan="2" align="right"><input type="submit" value="Add"
id="addBtn"/></td>
  </tr>
 </table>
</form>
```

APPENDIX C

Sauce Labs Code Examples

The manually written test scripts for Sauce Labs are available below. The first executes a simple test that opens the website and verifies its title in various different environments. The purpose of this test was to explore parallelization and gain familiarity with the tool. The second test script performs and verifies the basic functionality of the system. This test was only run in a single environment, but could easily be expanded to more. The script was limited to save testing minutes, especially in the Mac environments, where fewer minutes are provided each month.

Parallelization Test: WebDriverTest.java

```
/*************************************************************
 * WebDriverTest
 * -------------
 *
 * Simple test that just opens the website
 *     http://bfthesis-todo.appspot.com/
 * and verifies that the title is correct.
 *
 * Several environments are run in parallel. On the free
 * version, only two actually execute at the same time.
 *
 * Environments used in this example:
 *  Windows XP    / Firefox v26
 *  Windows 8     / Internet Explorer v10
 *  Mac 10.6     / Safari v5
 *  iPhone iOS 10.9 / iPhone Safari v7 (portrait orientation)
 *  Linux       / Chrome v30
 *  Android Tablet / Android v4.0 (portrait orientation)
 *
 * Displays tests in SauceLabs with the following name:
 *     webDriver[(session#)]: (browser) (version) (environment)
 *************************************************************/

package test.java.com.unit;

import com.saucelabs.common.SauceOnDemandAuthentication;
import com.saucelabs.common.SauceOnDemandSessionIdProvider;
import com.saucelabs.junit.Parallelized;
import com.saucelabs.junit.SauceOnDemandTestWatcher;
import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;

// Continued on next page.
```

```
import org.junit.rules.TestName;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.CapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

import java.net.URL;
import java.util.LinkedList;

import static org.junit.Assert.assertEquals;

@RunWith(Parallelized.class)
public class WebDriverTest implements SauceOnDemandSessionIdProvider {

 /* Authentication, which requires the username and access key
   * The access key is replaced with <access_key> for security in
   * this book
   */
  public SauceOnDemandAuthentication authentication = new
SauceOnDemandAuthentication(
      "bfloss", "<access_key>");

  private String browser;
  private String os;
  private String version;

  /**
   * JUnit Rule that marks the Sauce Job as passed/failed when the
   * test succeeds or fails. You can see the pass/fail status on
   * your [Sauce Labs test page](https://saucelabs.com/tests).
   */
  public @Rule SauceOnDemandTestWatcher resultReportingTestWatcher =
new SauceOnDemandTestWatcher(this, authentication);

  /**
   * JUnit Rule that will record the test name of the current test.
   * This is referenced when creating the {@link
   * DesiredCapabilities}, so the Sauce Job is created with the test
   * name.
   */
  public @Rule TestName testName = new TestName();

  /**
   * An object that contains the environment information.
   * @param os Operating system
   * @param version Version of browser
   * @param browser Web browser
   */

// Continued on next page.
```

```
  private WebDriverTest(String os, String version, String browser) {
    super();
    this.os = os;
    this.version = version;
    this.browser = browser;
  }

/**
   * Creates a linked list of environments that will be run
   * sequentially
   * @return LinkedList Environments
   * @throws Exception
   */
  @Parameterized.Parameters
  public static LinkedList browsersStrings() throws Exception {
    LinkedList browsers = new LinkedList();
    browsers.add(new String[]{Platform.XP.toString(), "26",
"firefox"});
    browsers.add(new String[]{Platform.WIN8.toString(), "10", "internet
explorer"});
    browsers.add(new String[]{Platform.MAC.toString(), "5", "safari"});
    browsers.add(new String[]{Platform.MAC.toString(), "7", "iphone"});
    browsers.add(new String[]{Platform.LINUX.toString(), "30",
"chrome"});
    browsers.add(new String[]{Platform.ANDROID.toString(), "4.0",
"android"});
    return browsers;
  }

  private WebDriver driver;
  private String sessionId;

  /**
   * The setup function sets the environment and creates the
   * RemoteWebDriver that connects to the user account by using
   * the account's username and access key. The access key was
   * changed to <access_key> in this example for security, but is
   * available on the Sauce Labs dashboard.
   */
  @Before
  public void setUp() throws Exception {
    // Setting up a variety of environments to test
    // parallelization and environmental offerings.
    DesiredCapabilities capabilities = new DesiredCapabilities();
    capabilities.setCapability(CapabilityType.BROWSER_NAME, browser);
    capabilities.setCapability(CapabilityType.VERSION, version);
    capabilities.setCapability(CapabilityType.PLATFORM,
Platform.valueOf(os));

// Continued on next page.
```

```
   // Settings for mobile devices only
    if(browser.equalsIgnoreCase("android"))
    {
      capabilities.setCapability("device-type", "tablet");
      capabilities.setCapability("device-orientation", "portrait");
    }
    else if(browser.equalsIgnoreCase("iphone"))
    {
      capabilities.setCapability("device-orientation", "portrait");
    }

    String name = testName.getMethodName() + ": " + browser + " " +
version + " " + Platform.valueOf(os);
    capabilities.setCapability("name", name);

    this.driver = new RemoteWebDriver(
        new URL("http://" + authentication.getUsername() + ":" + authen-
tication.getAccessKey() + "@ondemand.saucelabs.com:80/wd/hub"),
        capabilities);
    this.sessionId = ((RemoteWebDriver)driver).getSessionId().toString();
  }

  /**
   * Although not used directly, the session ID must be stored.
   */
  @Override
  public String getSessionId() {
    return sessionId;
  }

  @Test
  public void webDriver() throws Exception {
    // Simple test - just go to the site and verify the title.
    driver.get("http://bfthesis-todo.appspot.com/");
    assertEquals("To Do List", driver.getTitle());
  }

  @After
  public void tearDown() throws Exception {
    driver.quit();
  }
}
```

Functionality Test: TodoTest.java

```
/*************************************************************
 * TodoTest
 * -------------
 *
 * Series of basic tests covering the functionality of
 *     http://bfthesis-todo.appspot.com/
 *
 * Only one environment is being used - Windows 8 / Firefox 26.
 *
 * Basic functionality tested:
 *   Login
 *   Verify login
 *   Add ToDo
 *   Verify that a ToDo exists
 *   Add ToDo
 *   Verify that two ToDo items exist
 *   Verify data in table
 *   Mark ToDo items as complete
 *   Logout
 *
 * Displays in SauceLabs as
 *     todo
 *************************************************************/

package test.java.com.unit;

import com.saucelabs.common.SauceOnDemandAuthentication;
import com.saucelabs.common.SauceOnDemandSessionIdProvider;
import com.saucelabs.junit.SauceOnDemandTestWatcher;

import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TestName;
import org.openqa.selenium.By;
import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.ArrayList;
import java.util.List;

import main.java.com.book.todo.model.Todo;
import main.java.com.book.todo.dao.Dao;
import static org.junit.Assert.assertEquals;
// Continued on next page.
```

```
public class TodoTest implements SauceOnDemandSessionIdProvider {

  public SauceOnDemandAuthentication authentication = new
SauceOnDemandAuthentication(
      "bfloss", "<access_key>");

  /**
   * JUnit Rule that marks the Sauce Job as passed/failed when the
   * test succeeds or fails. You can see the pass/fail status on
   * your [Sauce Labs test page](https://saucelabs.com/tests).
   */
  public @Rule SauceOnDemandTestWatcher resultReportingTestWatcher =
new SauceOnDemandTestWatcher(this, authentication);

 /**
   * JUnit Rule that will record the test name of the current test.
   * This is referenced when creating the {@link
   * DesiredCapabilities}, so the Sauce Job is created with the test
   * name.
  */
  public @Rule TestName testName = new TestName();

  private WebDriver driver;
  private String sessionId;

  private Dao dao = Dao.INSTANCE;
  private List<Todo> todoList = new ArrayList<Todo>();

  @Before
  public void setUp() throws Exception {
    // Running on Windows 8 using Firefox v26.
    DesiredCapabilities capabilities = DesiredCapabilities.firefox();
    capabilities.setCapability("version", "26");
    capabilities.setCapability("platform", Platform.WIN8);
    capabilities.setCapability("name", testName.getMethodName());
    this.driver = new RemoteWebDriver(
        new URL("http://" + authentication.getUsername() + ":" +
            authentication.getAccessKey() +
"@ondemand.saucelabs.com:80/wd/hub"),
        capabilities);
    this.sessionId =
((RemoteWebDriver)driver).getSessionId().toString();
  }

 @Override
  public String getSessionId() {
    return sessionId;
  }

// Continued on next page.
```

```
  @Test
  public void todo() throws Exception {
    // Verify that we are at the correct webpage.
    driver.get("http://bfthesis-todo.appspot.com/");
    assertEquals("To Do List", driver.getTitle());

    // Logout, if necessary.
    try
    {
      if(driver.findElement(By.id("logoutLink")).isDisplayed())
      {
        driver.findElement(By.id("logoutLink")).click();
        driver.navigate().refresh();
      }
    }
    catch(Exception e)
    {

      // Not logged in.
    }

    // Verify that no user is logged in. Then log in as bfthesis.
    // No password is required in the application at this point.
    // driver.navigate().refresh() frequently used to verify data
    // persistence.
    assertEquals("", driver.findElement(By.id("user")).getText());
    driver.findElement(By.id("user")).sendKeys("bfthesis");
    driver.findElement(By.id("loginBtn")).submit();
    driver.navigate().refresh();

    // Verify that the proper user is being "welcomed" (they are
    // "logged in").
    WebElement tbl = driver.findElement(By.id("welcomeMessage"));
    List<WebElement> rows = tbl.findElements(By.tagName("tr"));
    List<WebElement> cols = rows.get(0).findElements(By.tagName("td"));
    assertEquals("bfthesis", cols.get(1).getText());

    // The values to set. yyyy-MM-dd is the calendar object's
    // default.
    String[][] setVals = new String[][]
    {
      {"Test", "First ToDo", "2015-01-01"},
      {"Test 2", "Second ToDo", "2014-12-31"}
    };

    // The values to verify. Need the current date for "Create
    // Date". The dates display in MM/dd/yyyy format.
    Date date = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy");
    String curDate = sdf.format(date);

// Continued on next page.
```

```
    String[][] getVals = new String[][]
    {
      {"Test 2", "Second ToDo", curDate, "12/31/2014"},
      {"Test", "First ToDo", curDate, "01/01/2015"}
    };

    // Adding two ToDo items. They will be added as 0, 1,
    // but will display as 1, 0 due to due date order.
    driver.findElement(By.id("name")).sendKeys(setVals[0][0]);
    driver.findElement(By.id("desc")).sendKeys(setVals[0][1]);
    driver.findElement(By.id("dueDt")).sendKeys(setVals[0][2]);
    driver.findElement(By.id("addBtn")).submit();
    driver.navigate().refresh();

    tbl = driver.findElement(By.id("todoCount"));
    rows = tbl.findElements(By.tagName("tr"));
    cols = rows.get(0).findElements(By.tagName("td"));
    assertEquals("1", cols.get(1).getText());

    driver.findElement(By.id("name")).sendKeys(setVals[1][0]);
    driver.findElement(By.id("desc")).sendKeys(setVals[1][1]);
    driver.findElement(By.id("dueDt")).sendKeys(setVals[1][2]);
    driver.findElement(By.id("addBtn")).submit();
    driver.navigate().refresh();

    // Verify that the table contains the correct information.
    // Skipping the "Done" column, but will use it in the next
    // test.
    tbl = driver.findElement(By.id("todoCount"));
    rows = tbl.findElements(By.tagName("tr"));
    cols = rows.get(0).findElements(By.tagName("td"));
    assertEquals("2", cols.get(1).getText());

    tbl = driver.findElement(By.id("todos"));
    rows = tbl.findElements(By.tagName("tr"));
    for(int i = 0; i < getVals.length; i++)
    {
      cols = rows.get(i+1).findElements(By.tagName("td"));
      for(int j = 0; j < getVals[i].length; j++)
      {
        assertEquals(getVals[i][j], cols.get(j).getText());
      }
    }

   // Class done includes marking the todo items as completed
    // AND logging out. Verify both.
    driver.findElement(By.className("done")).click();
    driver.navigate().refresh();
    tbl = driver.findElement(By.id("todoCount"));
    rows = tbl.findElements(By.tagName("tr"));
    cols = rows.get(0).findElements(By.tagName("td"));
    assertEquals("0", cols.get(1).getText());

// Continued on next page.
```

```
    assertEquals("", driver.findElement(By.id("user")).getText());
  }

  @After
  public void tearDown() throws Exception {
    driver.quit();
  }
}
```

References

1. C. Kaner, "Black Box Software Testing," Fall-2006. 1

2. B. Floss and S. Tilley, "Software Testing as a Service: An Academic Research Perspective," in *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, 2013, pp. 421–424. DOI: 10.1109/SOSE.2013.97. 2, 17, 18, 29

3. S. Tilley. *Hard Problems in Software Testing*. Online at www.hpst.net. 2, 6

4. S. Tilley, "Hard Problems in Software Testing," *SurveyMonkey*. [Online]. Available: http://www.surveymonkey.com/s/GTXKPJL. [Accessed: 26-Jan-2014]. 2

5. D. Firesmith, "Common Testing Problems: Pitfalls to Prevent and Mitigate," *SEI Blog*, 05-Apr-2013. [Online]. Available: http://blog.sei.cmu.edu/post.cfm/common-testing-problems-pitfalls-to-prevent-and-mitigate. [Accessed: 07-Nov-2013]. 1

6. D. Firesmith, "Common Testing Problems: Pitfalls to Prevent and Mitigate," *SEI Blog*, 06-Apr-2013. [Online]. Available: http://blog.sei.cmu.edu/post.cfm/common-testing-problems-pitfalls-to-prevent-and-mitigate-2. [Accessed: 07-Nov-2013]. 1

7. D. Firesmith, "Common Testing Problems: Pitfalls to Prevent and Mitigate," A1AA Case Conference, 12-Sep-2012. 1

8. D. Firesmith, "Common Testing Problem Checklists: Symptoms and Recommendations," CMU/SEI Technical Report, 10-Oct-2012. 1, 2

9. D. Firesmith, *Common System and Software Testing Pitfalls: How to Prevent and Mitigate Them: Descriptions, Symptoms, Consequences, Causes, and Recommendations*. Addison-Wesley Professional, 2013. 1

10. L. Riungu-Kalliosaari, "Leah Riungu-Kalliosaari - Finland | LinkedIn," *LinkedIn*. [Online]. Available: http://fi.linkedin.com/pub/leah-riungu-kalliosaari/10/34b/a22. [Accessed: 26-Jan-2014].

11. L. M. Riungu, O. Taipale, and K. Smolander, "Software Testing as an Online Service: Observations from Practice," in *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, 2010, pp. 418 –423. DOI: 10.1109/ICSTW.2010.62. 2, 30, 31

12. L. M. Riungu, O. Taipale, and K. Smolander, "Research Issues for Software Testing in the Cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 557 –564. DOI: 10.1109/CloudCom.2010.58. 2, 31

13. L. Riungu-Kalliosaari, O. Taipale, and K. Smolander, "Testing in the Cloud: Exploring the Practice," *IEEE Software*, vol. 29, no. 2, pp. 46–51, Mar. 2012. DOI: 10.1109/MS.2011.132. 2, 30, 31

14. J.M. Corbin and A. Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 3rd Edition. Thousand Oaks, California: Sage Publications, 2007. 6

15. P. Mell and Grance, Timothy, "The NIST Definition of Cloud Computing." National Institute of Standards and Technology, Sep-2011. 15

16. J. Treadwell, "Open Grid Services Architecture: Glossary of Terms." Hewlett-Packard, 12-Dec-2007. 18

17. G. Candea, S. Bucur, and C. Zamfir, "Automated software testing as a service," in *Proceedings of the 1st ACM symposium on Cloud computing*, New York, NY, USA, 2010, pp. 155–160. DOI: 10.1145/1807128.1807153. 18, 19, 21, 31, 73

18. L. Yu, L. Zhang, H. Xiang, Y. Su, W. Zhao, and J. Zhu, "A Framework of Testing as a Service," in *Management and Service Science, 2009. MASS '09. International Conference on*, 2009, pp. 1 –4. DOI: 10.1109/ICMSS.2009.5302717. 19, 25

19. S. K. K. Priyadarsini, V. Balasubramanian, "Cloud Testing as a Service," *Int. J. Adv. Eng. Sci. Technol. IJAEST*, vol. 6, no. 2, pp. 173–177, 2011. 19

20. L. Yu, W.-T. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, and W. Zhao, "Testing as a Service over Cloud," in *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, 2010, pp. 181 –188. DOI: 10.1109/SOSE.2010.36. 19

21. S. Huang, Z. J. Li, Y. Liu, and J. Zhu, "Regression Testing as a Service," in *SRII Global Conference (SRII), 2011 Annual*, 2011, pp. 315 –324. DOI: 10.1109/SRII.2011.105. 22

22. M. Yan, H. Sun, X. Wang, and X. Liu, "Building a TaaS Platform for Web Service Load Testing," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, 2012, pp. 576–579. DOI: 10.1109/CLUSTER.2012.20. 23

23. M. Yan, H. Sun, X. Wang, and X. Liu, "WS-TaaS: A Testing as a Service Platform for Web Service Load Testing," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, 2012, pp. 456–463. DOI: 10.1109/ICPADS.2012.69. 24, 28

24. N. Kosmatov, N. Williams, B. Botella, and M. Roger, "Structural Unit Testing as a Service with PathCrawler- online.com," *2013 IEEE Seventh Int. Symp. Serv.- Oriented Syst. Eng.*, pp. 435–440, 2013. DOI: 10.1109/SOSE.2013.78. 24, 73

25. W.-T. T. Jerry Gao, Xiaoying Bai, "Cloud Testing - Issues, Challenges, Needs and Practice," *Softw. Eng. Int. J. SEIJ*, vol. 1, no. 1, pp. 9–23, 2011. 30, 31

26. O. Starov and S. Vikomir, "Integrated TaaS Platform for Mobile Development: Architecture Solutions," in *Automation of Software Test(AST), 8th International Conference on*, 2013, pp. 1–7. DOI: 10.1109/IWAST.2013.6595783. 31

27. R. Abdullah, "Toward developing software testing as a service (staas) model in cloud computing: a case of collaborative knowledge management system," in *Proceedings of the 11th WSEAS international conference on Software Engineering, Parallel and Distributed Systems, and proceedings of the 9th WSEAS international conference on Engineering Education*, Stevens Point, Wisconsin, USA, 2012, pp. 25–29. 32

28. J. R. Haryadi S. Gunawi, Thanh Do, Joseph M. Hellerstein, Ion Stoica, Dhruba Borthakur, "Failure as a Service (FaaS): A Cloud Service for Large-Scale, Online Failure Drills," University of California at Berkley, 2011. 33

29. "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 61012-1990*, pp. 1–84, 1990. DOI: 10.1109/ieeestd.1990.101064. 34

30. "Software and systems engineering Software testing Part 1: Concepts and definitions," *ISOIECIEEE 29119-12013E*, pp. 1–64, Sep. 2013. 34

31. "Software and systems engineering Software testing Part 2: Test processes," *ISOIECIEEE 29119-22013E*, pp. 1–68, Sep. 2013. DOI: 10.1109/IEEESTD.2013.6588543. 34

32. "Software and systems engineering Software testing Part 3: Test documentation," *ISOIECIEEE 29119-32013E*, pp. 1–138, Sep. 2013. DOI: 10.1109/IEEESTD.2013.6588540. 34

33. J. Tian, *Software Quality Engineering*, 1st Edition. Hoboken, New Jersey: Wiley-IEEE Computer Society Press, 2005. DOI: 10.1002/0471722324.part1. 34

34. "Localization Testing," *Microsoft MSDN*. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa292138(v=vs.71).aspx. [Accessed: 30-Jan-2014]. 23

35. E. Morris, W. Anderson, S. Bala, D. Carney, J. Morley, P. Place, and S. Simanta, "Testing in Service-Oriented Environments," Software Engineering Institute, Hanscom AFB, MA, CMU/SEI-2010-TR-011, Mar. 2010. 36

36. "Sauce Labs," *Sauce Labs: Selenium Testing, Mobile Testing, JS Unit Testing and More*. [Online]. Available: https://saucelabs.com/. [Accessed: 02-Feb-2014]. 39

37. "Selenium Builder," *sebuilder - GitHub*. [Online]. Available: https://github.com/se-builder/se-builder. [Accessed: 02-Feb-2014]. 40

38. "SOASTA," SOASTA | *Load and Performance Testing for Web and Mobile Apps*. [Online]. Available: http://www.soasta.com/. [Accessed: 02-Feb-2014]. 41

39. "BlazeMeter," *BlazeMeter: JMeter Load Testing Cloud*. [Online]. Available: http://blazemeter.com/. [Accessed: 02-Feb-2014]. 42

40. D. Rosenberg, "BlazeMeter raises funds for cloud-based load testing," *CNET*, 06-Dec-2011. 42

41. "App Engine - Google Cloud Platform." [Online]. Available: https://cloud.google.com/products/app-engine/. [Accessed: 05-Feb-2014]. 46

42. "The Admin Console - Google App Engine." [Online]. Available: https://developers.google.com/appengine/docs/adminconsole/. [Accessed: 05-Feb-2014]. 46

43. L. Vogel, "Google App Engine Tutorial for Java," *Vogel/a*, 03-Dec-2010. [Online]. Available: http://www.vogella.com/tutorials/GoogleAppEngineJava/article.html#todo. [Accessed: 15-Jan-2014]. 46, 81

44. "Sauce Labs," *Sauce for Selenium Builder*. [Online]. Available: https://saucelabs.com/builder. [Accessed: 02-Feb-2014]. 47

45. "Mozilla, creator of Firefox and Firefox OS, runs tests quickly and reliably with Open Sauce." [Online]. Available: http://saucelabs.com/downloads/Mozilla-Sauce_Labs_Case_Study.pdf. [Accessed: 02-Feb-2014]. 53

46. "Okta Increases Developer Productivity by 80% Using Sauce's Testing Cloud." [Online]. Available: https://saucelabs.com/downloads/Okta-Sauce_Labs_Case_Study.pdf. [Accessed: 02-Feb-2014]. 54

47. "Eventbrite Catches Issues Sooner and Debugs Faster Using Sauce Labs For Its Billion-Dollar Event Management Platform." [Online]. Available: http://saucelabs.com/downloads/Eventbrite-Sauce_Labs_Case_Study.pdf. [Accessed: 02-Feb-2014]. 54

48. "2012 Summer Olympic Games - SOASTA," *SOASTA*. [Online]. Available: http://www.soasta.com/customers/case-studies/2012-summer-olympic-games-2/. [Accessed: 07-Feb-2014]. 61

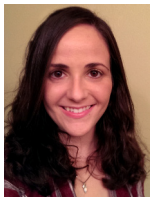
49. "Microsoft Performance Tests Office.com, its Customer-facing Web Site - SOASTA," *SOASTA*. [Online]. Available: http://www.soasta.com/customers/case-studies/microsoft-performance-tests-office-com-its-customer-facing-web-site/. [Accessed: 07-Feb-2014]. 61

50. "KentuckyDerby.com Crosses the Finish Line," *SOASTA*. [Online]. Available: http://www.soasta.com/customers/case-studies/kentuckyderby-com-crosses-the-finish-line/. [Accessed: 07-Feb-2014]. 62

51. "Testimonials from Satisfied BlazeMeter Customers | BlazeMeter," *BlazeMeter: JMeter Load Testing Cloud*. [Online]. Available: http://blazemeter.com/testimonials. [Accessed: 07-Feb-2014]. 68

Author Biographies

Scott Tilley is a Professor in the Department of Education and Interdisciplinary Studies at the Florida Institute of Technology, where he is Director of Computing Education. He is Chair of the Steering Committee for the IEEE Web Systems Evolution (WSE) series of events and a Past Chair of the ACM's Special Interest Group on Design of Communication (SIGDOC). He is an ACM Distinguished Lecturer. His current research focuses on software testing, cloud computing, and system migration. He writes the weekly "Technology Today" column for the *Florida Today* newspaper (Gannett).



Brianna Floss is a software developer at Lockheed Martin. She previously worked in software testing at several companies. Early results from her research on software testing challenges and the capabilities and potential of Testing as a Service were presented at the *IEEE 7th International Symposium on Service-Oriented System Engineering* (SOSE 2013). She holds an MSE from the Florida Institute of Technology.