TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABREVIATIONS

ETS	École de Technologie Supérieure
SWEBOK	The Guide to the Software Engineering Body of Knowledge
ISTQB	International Software Testing Qualifications Board
CAST	Computer Aided Software Testing
API	Application Program Interface
XML	Extensible Markup language

INTRODUCTION

Context

Software testing is an important software engineering process which is tasked with the evaluation of the quality of a software product. Its impact on the lifecycle of a software product is significant and estimated at around 40% of the total development time (Laporte et al., 2012). Courses on software testing are mandatory in many, if not most, software engineering courses. In particular, testing is featured as one of the main Knowledge Areas in the Software Engineering Body of Knowledge (SWEBOK) (Bourque and Fairley, 2015), which is a consensus document around which many SE programs, including that of ETS, is modeled. On a more practical / industry-oriented side, there are many software qualification organizations that specialize on testing; the biggest one being the International Software Testing Qualification Board (ISTQB, 2016), which operates over 117 countries world-wide. In short, there is considerable interest in learning various aspects, processes and technologies of testing.

A key factor to the renewed push for systematic and serious testing resides in the large choice of tools, testing tools, that help in quality control by automating various testing activities and reducing human errors. These tools support the testing process and assist in the production of high-quality software products, with reduced time and cost. The software industry is one with high velocity, in terms of new paradigms and technologies. This dynamic applies also to supporting tools such as testing tools, which have to evolve accordingly and propose new techniques, or refine older ones. As software written to test software products, testing tools must evolve in conjunction with the testing needs of newer technologies. Updates and new releases are thus key to the relevance of testing tools and they are (or should be) documented with meaningful and clear terminology.

New release documentation includes many items for different audiences. User manuals are important but in the case of engineering tools destined to individuals with above average technicality, artifacts such as change logs or release notes become a privileged way for tool developers to communicate about the changes and/or new features they introduced in their tools. Release notes describe new capabilities, known problems, and platform requirements that are necessary for a proper product operation (Abran et al., 2004). They are usually intended for end-users and contain useful instructions on how to install or upgrade to the current package. change logs tend to be a bit more technical and are generally presented as curated and chronologically ordered lists of meaningful changes per version of a software[1]. They are more useful from a developer/tester perspective.

In short, change logs and release notes are important release documentation that communicate, in a concise way, important information on a software and its evolution. We believe they provide good insights, with tolerable noise level, both on the services offered by a software and/or the technologies it depends on or is relevant for. The central question of this thesis is thus related to the adequacy of the terminologies used to teach software testing (e.g., SWEBOK, ISTQB) with respect to the one used by developers of testing tools to communicate about their products.

Research on change logs and release notes is relatively new, and thus, more research work is needed to offer insights into these artifacts. In the literature, several works were conducted to identify the challenges involved in writing both release notes and change logs. Guidelines and styles were proposed to write better release notes (Abebe et al., 2016) and change logs (GNU; Hutterer, 2009; Lacan, 2016). Machine learning techniques have been proposed to identify important issues from one release to another and then, generate automatically change logs and release notes (Abebe et al., 2016; Moreno et al., 2014, 2016).

Text mining is a process to automatically extract data from different written resources in order to discover unknown information. It can be used to study change logs and release notes for a

[1] http://keepachangelog.com/

better understanding of software maintenance. Text mining approaches have been proposed to understand software maintenance from the perspective of change logs and release notes (Yu, 2009). (Garousi and Felderer, 2016) presented a survey collecting recent papers from industry conferences and research conferences on software testing and presented word cloud images of common phrases used in collected papers spanning across two domains (industry and research). The purpose of their efforts was to find keywords from word clouds and to find the intersection between industry-academic materials. An important idea behind this effort is to address the lack of collaboration in software testing between industry and academia in terms of communication about challenges. In this context, it is necessary to carry out empirical studies and analyze terms used in change logs and release notes, so that the terminology of software testing tools can be understood better.

In the current work, we are interested in analyzing change logs and release notes of software testing tools, in an effort to find how well they communicate on their important issues. Our analysis is carried out using i) change logs and release notes from a sample of testing tools and ii) terms from established sources like *The guide to the Software Engineering Body of Knowledge (SWEBOK) V3* and *International Software Testing Qualifications Board (ISTQB) standard glossary of terms v3.1*.

Objectives

The main objective of our research is to analyze the change logs and release notes of software testing tools to get insights on their terminologies. In particular, we aim to conduct a study of change logs and release notes to see how well they reflect the terms of established references (SWEBOK V3, ISTQB standard glossary 3.1).

To achieve these objectives, we set out to answer the following research questions:

a. How much of the terms from established sources (SWEBOK v3, ISTQB v3.1) related to testing are reflected in change logs and release notes?

b. Which would be the dominant terms in a terminology extracted from change logs and release notes?

c. What insights can we get by comparing ISTQB, SWEBOK and Log based terminologies?

Thesis Structure

The remainder of the thesis is organized as follows: Chapter 1 presents background concepts and literature review relevant to our work. Chapter 2 describes our methodology: data collection and execution. Chapter 3 presents our results: raw numbers and discussion, threats to validity and the limits of the approach. Finally, this thesis ends with a conclusion and future perspectives of our research work.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we present the concepts that are relevant for our research study. We begin by presenting and defining software testing. Then, we present the very few relevant works on software testing carried out in academia and industry. In the final section, we discuss proposed approaches for the study of change logs and release notes (release documentation).

1.1 Software Testing

Software testing is an integral part in the software development life-cycle. Inadequate software testing leads to major risks and consequences (Garousi and Zhi, 2013). The American National Institute of Standards and Technology (NIST) 2002 report stated that lack of infrastructure in software testing alone cost $62 billion USD per year in United States (Tassey, 2002). The main goal of software testing is to find defects and faults in every stage (the earlier the better) of the development of a software. This is a necessary process as human mistakes are inevitable, and considering that costs to fix these faults when the software is in maintenance are expensive. With technology playing nowadays a pivotal role in human life's (transport, health, etc.), a defect in a software system may lead to disaster. For example, a testing information systems failure resulted in a death in London Ambulance Service software (Finkelstein and Dowell, 1996). In this context, software testing tools are deployed to automate the test process (Emami et al., 2011) in order to improve quality and efficiency of the product, and reduce testing related costs.

1.1.1 Definition of Software Testing

In the literature, several definitions have been given to software testing. The guide to the Software Engineering Body of Knowledge(SWEBOK v3) defines Software Testing as, the 'dynamic verification' that a program provides 'expected behaviours' on a 'finite' set of test cases (Bourque et al., 2014). These test cases are suitably selected from the unusually ’infinite execution domain’.

According to **ANSI/IEEE 1059 standard**: Testing can be defined as a process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item (IEEE, 1994).

The Art of Software Testing - Second Edition defines: Software testing [as] a process, or a series of processes, designed to make sure computer code does what it was designed to do and that it does not do anything unintended (Reid, 2005).

Finally, **The Art of Software Testing - Third Edition** defines: Testing [as] the process of executing a program with the intent of finding errors. (Myers et al., 2011).

1.1.2 Software Testing Tools

The IEEE Standard Glossary of Software Engineering Terminology defines a software tool as: "a computer program used in the testing, development, maintenance, or analysis of a program or its documentation" (Radatz et al., 1990). These include cross-reference generators, time analyzers, flow-charters, decompilers, test case generators, etc. There is a wide variety of tools available today to provide assistance in every phase of the testing process. There are no universal testing tools that would cater to all testing needs of all levels and phases of a software development cycle. Rather, testing tools can be categorized in a number of ways:

a. The testing project or activity in which they are employed (e.g., code verification, test planning, test execution);

b. The descriptive keyword, i.e. the specific function performed by the tool (e.g., capture/replay, logic coverage, compactor);

c. A major area of classification going beyond testing only (e.g., test *management*, *static analysis*, simulator).

As an example, we can cite the categorization of (Abran et al., 2001):

a. Test generators

b. Test execution frameworks

c. Test evaluation tools

d. Test management tools

e. Performance analysis tools

While many tools are useful mainly in keeping track and managing tests that are scheduled or done, some testing tools provide automation in core testing activities. This reduces manual testing, which is costly, time consuming and error prone (Bajaj, 2014). Test automation helps in finding issues that are overlooked or not re-verified (regression) (Bajaj, 2014). However, a recent survey in (Garousi and Zhi, 2013) reveals that about 80 percent of the Canadian firms use manual testing due to acquisition costs and/or lack of training for good quality test automation tools.

1.1.3 Academic work on software testing and software testing tools

There are several academic works carried out on testing tools. A very recent (2016) study (Garousi et al., 2016) finds that the level of joint industry-academic collaboration in software testing is very low. Researchers have few insights on problems that are important to practitioners, while practitioners fail to learn what researchers have already discovered that might be useful to them. In fact, industrial problems are often devoid of scientific challenges, while at the same time more complex than the mostly small projects for which academia develops rigorous, but hardly scalable, solutions. A recent survey of 3000 employees in Microsoft suggests that top-cited research articles in Software Engineering (SE) are not relevant or useful to their everyday challenges (Lo et al., 2015).

A 2013 survey on software testing practices in Canada (Garousi and Zhi, 2013) identified current trends and challenges in testing, in an effort to provide a window into industry practices and encourage more academia–industry collaborations. Some of the notable findings of this survey are as followed:

- Canadian firms are giving more importance to testing related training;
- More effort and attention is spent on unit testing and functional testing;
- When it comes to testing tools, web application and NUnit tools overtook IBM rational tools and JUnit;
- Many of the companies are still using the Test Last Development (TLD) and only a small number companies trying to implement new development approaches such as Test Driven Development (TDD) and Behaviour Driven Development (BDD);
- Canadian firms are considering using new techniques, such as the mutation testing;
- A majority of Canadian firms use a combination of two coverage metrics: condition and decision coverage;
- In most Canadian companies, testers are out-numbered by developers, with ratios ranging from 1:2 to 1:5;

An older survey (Emami et al., 2011) of 152 open source software testing tools finds that the majority of the tools available were meant for performance testing (22%) and unit testing (21%). On the other hand, only 3% were useful for test management and database testing. Based on that survey, JAVA is the most supported programming platform by open source software testing tools. Almost 39% of testing tools support JAVA programming platform. It is especially notable that no other programming platform is supported by even as low as 10% of the tools. On the other hand, Visual Basic and database are the least supported languages/-concepts for the testing tools surveyed. When it comes to open source software testing tools, concerns about their maintenance are real, due to the non-profit and community driven natures of these initiatives. The survey found that 77% of the surveyed open source tools have had an update (version release) within the last six months.

1.1.4 Software testing terminology

Clear and unambiguous communication, through a shared terminology, is particularly helpful in any industry. A common understanding of terms allows people to communicate ideas more rapidly with less need for lengthy explanations. This is relevant in software testing as well, in many somewhat subtle ways. For example, testing can reveal *failures*, but only *faults* can and must be removed (Abran et al., 2004). In the literature, there are several works which present and discuss testing related terminology. The initial version of the SWEBOK trial version Abran et al. (2001) explicitly addresses and discusses testing-related terminology. The same applies to (Utting et al., 2006), which tries to describe and define testing related terminology. From practitioners perspective, there is the ISTQB standard glossary (ISTQB, 2016), which include three levels of terminology (foundation, advanced and expert).

1.2 Release documentation (change logs and release notes)

Release documentation takes many forms and goes from user-friendly manuals and tutorials to more technically oriented change logs. While very useful, user documentation tends to be scattered (pages of a website, plus videos, plus pdfs) and may not reflect as much as change logs and release notes information pertaining to features that are still be worked on, bugs that have been fixed etc. Unlike these, change logs and release notes, which are sometimes treated the same by developers, are often consolidated in a single, exploitable document, making them candidates of choice for studies like ours.

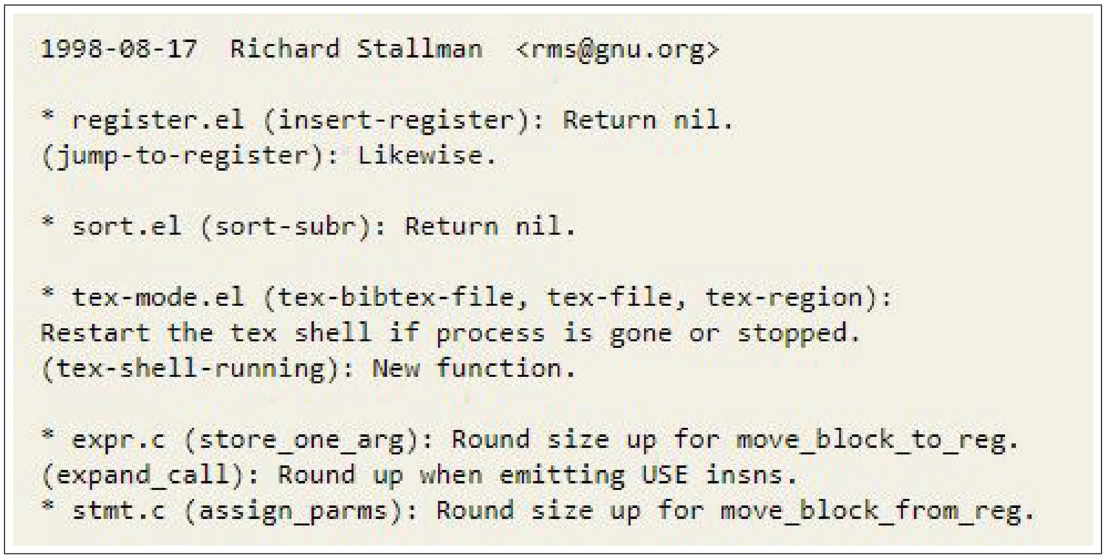
1.2.1 change logs

A change log is a file where all changes of a software system between two different releases are stored, in chronological order (Chen et al., 2004). In general, change logs are targeted at developers, testers and maintainers. They may provide detailed explanation or information about a change, its location(s), its creator's name and contact, dates, etc. As such, these documents represent a valuable source of data for empirical research on software maintenance (Chen et al., 2004). Most projects maintain their change log in a file simply named ChangeLog. Some other projects use HISTORY.txt, HISTORY.md, History.md, NEWS.txt, NEWS.md, News.txt, RELEASES.txt, RELEASE.md, releases.md, etc.

Unfortunately, there are no universally accepted guidelines and format on how to write change logs . There are, however, a few prominent online blogs/websites like GNU changelog (GNU) and *Keep a changelog* (Lacan, 2016) that share their ideas on both content and format of change logs . Their take is that change logs should contain, after each release, features that are added, bugs that are fixed, changes to the existing functionality, features that are deleted and features that are yet to be released. The writing style of a changelog suggested by (GNU) is presented in Figure 1.1. Though not technically part of a change log, semantic versioning (in short X.Y.Z for Major.Minor.Patch) is recommended as an effective way to communicate changes to the users of a software (Preston-Werner, 2013).

1.2.2 Release Notes

Release notes are considered one of the important *software trails* by (German, 2006). They contain essential information about a software product and its current release: what is new, changed, got fixed etc. Release notes are distributed with a software when it is made available for public use (Yu, 2009; Abebe et al., 2012). Much like change logs , they are used by developers to communicate with their software users about new changes/ updates in a software system, but arguably with less emphasis on technical aspects.



```
1998-08-17  Richard Stallman  <rms@gnu.org>

* register.el (insert-register): Return nil.
(jump-to-register): Likewise.

* sort.el (sort-subr): Return nil.

* tex-mode.el (tex-bibtex-file, tex-file, tex-region):
Restart the tex shell if process is gone or stopped.
(tex-shell-running): New function.

* expr.c (store_one_arg): Round size up for move_block_to_reg.
(expand_call): Round up when emitting USE insns.
* stmt.c (assign_parms): Round size up for move_block_from_reg.
```

Figure 1.1 Figure showing an example of change log format taken from Emacs and GCC (GNU)

1.2.3 Research on release documentation

Research on either change logs or release notes are mostly along two axes: i) (semi-)automatic generation of these documents, and ii) empirical studies based on them.

On automatic generation, we can cite (Buse and Weimer, 2010) which leverages commit logs for changes in source code into automatically documenting meaningful program changes, and thus reducing human errors (mostly in terms of missing entries in the change logs) (Buse and Weimer, 2010). A few years later, (Moreno et al., 2013) first proposed fine-granularity techniques to automatically generate human readable summaries for java classes, then the same authors presented, in (Moreno et al., 2014), an approach named ARENA (Automatic RElease Notes generAtor) to generate the release notes automatically. The tool extracts the information from source code files, summarizes it, and integrates it with information from versioning systems and issue trackers. According to the authors, the resulting logs are good approximations of the ones produced by the developers, with the added benefit that they include important information which is missing from the manually created release notes. Abebe et al. (2016) report the same kind of result for their approach. They carried out a study on the release notes of 15 different software systems to understand the type of information in release notes. Their study identifies six different types of information such as caveats, new features, bugs fixed and improvements. The authors found that most of the content in release notes is related to issues that have been addressed; though not all issues that have been addressed are included. Also, the studied systems accounted for 3 styles of release notes' writing: consolidated summary of selected issues, list of selected issues and list of all issues. Finally, the authors used machine learning techniques trained on 8 release notes of 3 different systems, to automatically suggest, with reasonable success, issues that needed to be listed in the release notes.

Empirical studies on release documentation include (Khomh et al., 2012), which proposes a case study of Mozilla Firefox to evaluate whether faster releases improve quality of software. The authors concluded that 6 week shorter releases offered better marketing opportunities to companies. Other studies such as (German, 2006) focus on visualisation of release notes and change logs to highlight which files are changed and who tend to change those files in an effort to identify the development stage of a project in a given time. Also notable is the vast body of research work on defect prediction models that are based on historical data and sometimes use change logs /release notes (He et al., 2012). Finally, and closer to our purposes, (Yu, 2009) used a keyword based approach to mine change logs of Linux and release notes of FreeBSD to extract useful software maintenance and evolution information. The authors concluded that content in the studied change logs is relatively accurate in representing the software changes.

CHAPTER 2

METHODOLOGY

In this chapter, we present the methodology and execution plan of our study. Figure 2.1 presents an overview of our methodology to answer our research questions on software testing terminology, in particular the extent to which concepts discussed in established learning resources for software testing are actually mentioned (or implemented) in software testing tools. First, we selected the established learning resources: SWEBOK, and ISTQB (*S1*) and the set of testing tools (*S2*). Then, we extracted a terminology from each of the learning resources (*T1, T2*) and collected the release documentation (change logs or release notes) of the tools (*L1*). After which, we processed said documentation to eliminate content not related to the testing functionalities offered by the tools, resulting in "trimmed logs" (*L2*). Subsequently, we extract from the trimmed logs a new terminology, which we refer to as *Tool Terminology*. These steps are the basis on which we investigated answers to our three research questions.

2.1 The learning resources: SWEBOK v3, ISTQB standard glossary v3.1

We selected two well-established learning resources: SWEBOK v3 as the more theoretical academia-oriented resource and ISTQB v3.1 as the more practitioner-oriented resource. The Software Engineering Body of Knowledge (SWEBOK) is an international standard (under ISO/IEC TR 19759:2015) which makes an effort to bring a consistent view on software engineering worldwide. It is the end result of diverse contributions, insights and comments provided by many software engineering teachers and researchers on the main knowledge areas that define software engineering. The SWEBOK is a well known and established document that is used to structure undergraduate software engineering programs in many universities, such as ETS. We choose it to represent standard academia terminology for software testing. To do so, we manually went through the 16 pages of the chapter devoted to software testing (Chapter 4) and extracted a set of words related to software testing. Figure 2.2 presents an example of our

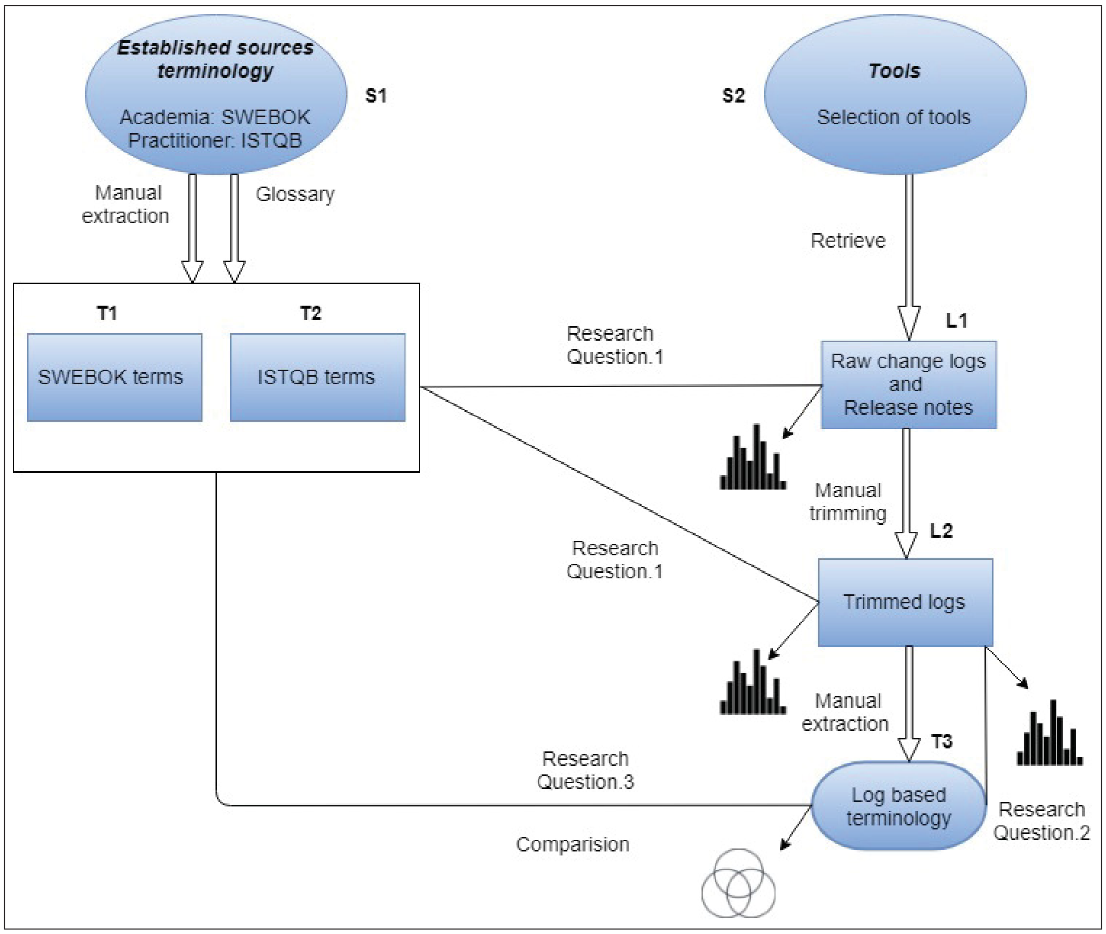
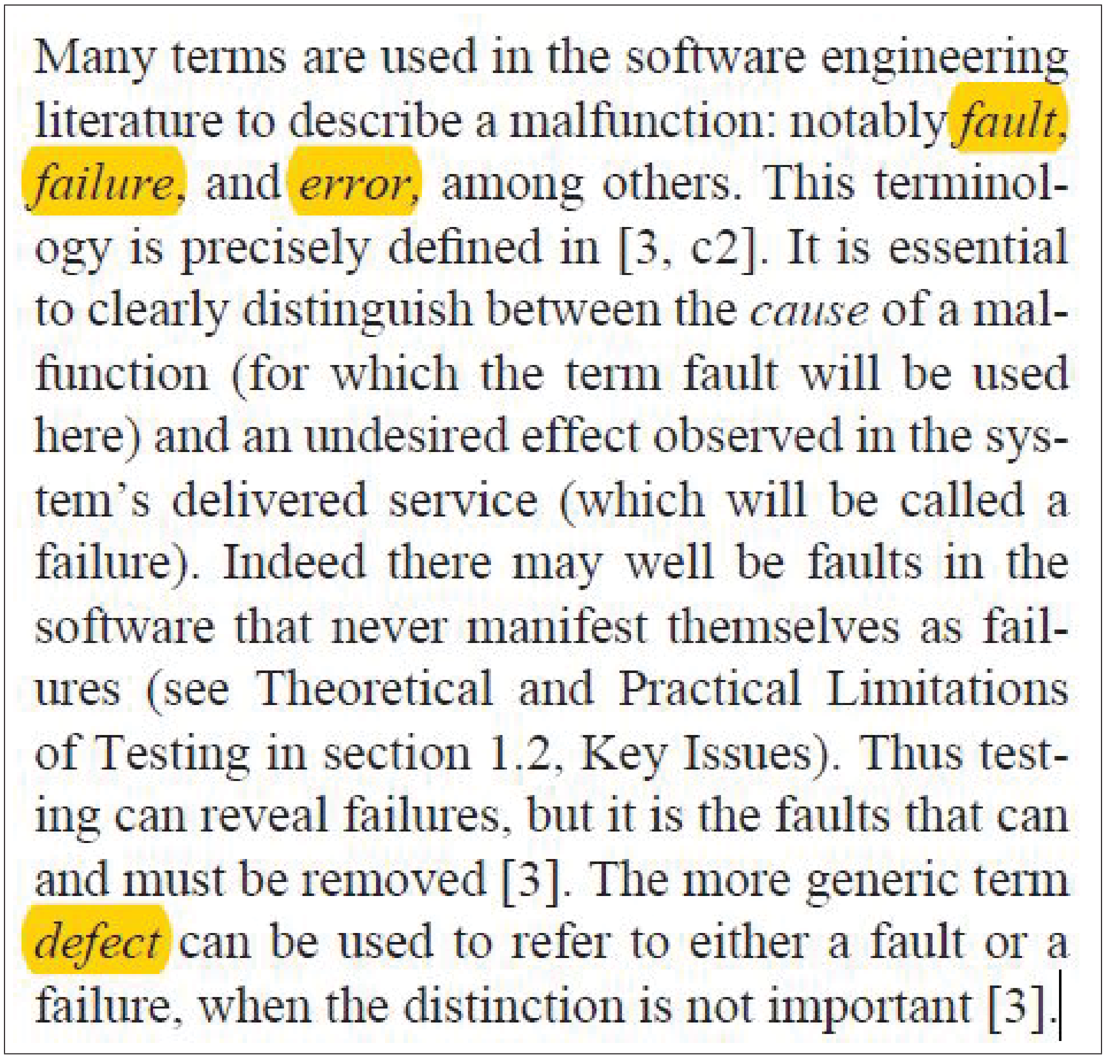


Figure 2.1 Flowchart depicting the empirical study

extraction of testing related terms from SWEBOK v3. At the end of this process, we retrieved 142 terms, such as *decision table*, *equivalent partition*, *defect* etc.

The International Software Testing Qualifications Board(ISTQB) is a software testing qualification certification organisation founded in 2002. As of June 2017 [1], it consists of 58 (national or regional) board members, such as the Canadian Software Testing Board (CSTB), and directly covers 82 countries. It has issued over 500,000 certifications, categorised in increasing levels of learning objectives (Foundation, Advanced, Expert) and modules (Core, Agile, Specialist). ISTQB's standard glossary lists concepts, terms and definitions to facilitate communication in (software) testing and related disciplines. For our study, we use its latest version:

[1] https://www.istqb.org/about-as/facts-figures.html



Many terms are used in the software engineering literature to describe a malfunction: notably *fault*, *failure*, and *error*, among others. This terminology is precisely defined in [3, c2]. It is essential to clearly distinguish between the *cause* of a malfunction (for which the term fault will be used here) and an undesired effect observed in the system's delivered service (which will be called a failure). Indeed there may well be faults in the software that never manifest themselves as failures (see Theoretical and Practical Limitations of Testing in section 1.2, Key Issues). Thus testing can reveal failures, but it is the faults that can and must be removed [3]. The more generic term *defect* can be used to refer to either a fault or a failure, when the distinction is not important [3].

Figure 2.2 Example of extraction of testing terms (highlighted in the text) from SWEBOK v3.
Taken from SWEBOK v3 (Bourque and Fairley, 2015).

the *Standard Glossary of Terms used in Software Testing Version 3.1*. We collected all the 725 terms into a text file from a PDF file provided by ISTQB. Terms present in the glossary include *condition coverage*, *memory leak*, *capture/playback* etc.

2.2 Tools and release documentation

Getting a representative sample for testing tools is a task for which there are no clear and established ways. Publications about software testing tools are rare and do not provide much information (Neto et al., 2012), and there is no established repository for listing testing tools. Therefore, we had to search for testing tools using mainly Wikipedia, as it is the top crowd-sourced encyclopedia, and Google, as the top search engine [2]. For Wikipedia, we started with the Wikipedia page for software testing tools (Wikipedia, 2015), which lists both testing tools and categories of software testing tools (such as GUI tools, load testing tools, security testing tools and unit testing tools). As for Google, we used the following query "software testing tools" and went through the first 20 pages [3] of search results, collecting in the process both links to testing tools and links to listings of testing tools, such as (QATestingTools.com).

We shortlisted 182 tools for our study, taking into account factors such as the status (still actively maintained or not) and source of the tool to its information. However, most of the tools in that initial list could not be included in our study because their change logs or release notes were unavailable. For instance, many of these tools were commercial tools such as HP Unified Functional Testing (UFT), IBM Rational Tool for which change logs or release notes were not made publicly available. We reached out to tool developers in many other cases by e-mail to request release information, but positive replies were relatively low: around 37%. Overall, out of 182 tools, we managed to get change logs and release notes information for 32 tools. These change logs and release notes are collected from repositories like GitHub, SourceForge and respective websites of the software testing tools. Tables 2.1 and 2.2 summarily present the 32 software testing tools along with their target language when relevant (usually for white box testing tools) and their category. They include very well-established testing tools such as Jest, JMeter, JUnit, Ranorex, Selenium, Sikuli, SoapUI, etc.

2.2.1 Standard format for the logs

The collected change logs and release notes are in various formats; an example is shown in Figure 2.3. In order to process these log files properly, we first arranged the version release information in chronological order of the release dates. Then, we added a deliminator at the be-

[2] in October 2017, its market share was estimated at 80.6%(marketshare, 2017)

[3] We stopped at 20 pages because there was then less and less new tools returned.

Table 2.1 Lists of tools selected for this study

Tool	Open source	Programming language	Testing category
Canoo	Yes	Javascript	Functional testing for web apps automated testing.
Easymock	Yes	Java	Open source testing framework for Java(unit testing tool)
FitNesse	Yes	Java,C++, Python and etc.	Acceptance testing framework
FunkLoad	Yes	Python	Web load testing, stress testing, and functional testing tool
Jest	Yes	JavaScript	Open source testing framework for JavaScript to test web applications, node.js services, mobile apps, and APIs.
Jmeter	Yes	Java	Performance and load testing
Junit	Yes	Java	Unit testing framework for java applications
Katalon Studio	Yes	Groovy, Java	Automation tool for web and mobile app testing.
Load Impact	No	Lua	Load testing tool
LoadStrom	No	N/A	Performance and load testing tool for cloud web applications.
MochaJS	Yes	JavaScript	JavaScript test framework
Mockito	Yes	Java	Open source testing framework for Java
NeoLoad	No	C, java	Load and stress testing tool to measure the performance of web and mobile applications
Qmetry	No	N/A	Test management tool
Ranorex	No	C, VB, .NET	GUI test automation framework
Robotium	Yes	Java	Android test automation framework
Sahi	Yes/No	Sahi Script	Web application testing
Selenium	Yes	C, Java, Python, PHP, Ruby, NodeJS, Groovy, Perl, Scala	Framework for web applications

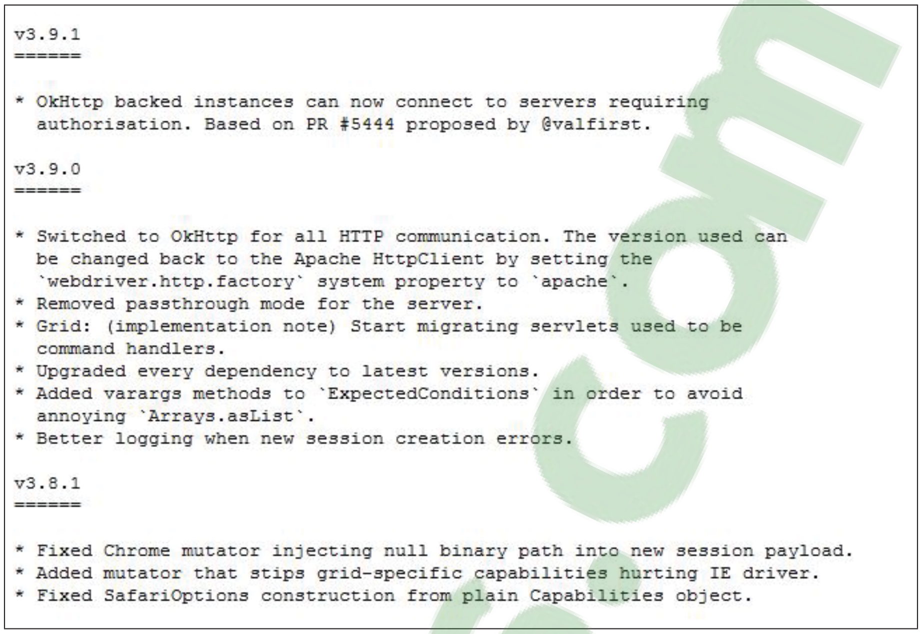
ginning and end of each version release to facilitate automatic parsing; resulting in an uniform format, as presented in Figure 2.4.

Table 2.2 Lists of tools selected for this study

Tool	Open source	Programming language	Testing category
Sikuli	Yes	Java, Python, C++	GUI testing tool
SoapUI	Yes	Groovy	Functional testing tool
Test Studio	No	C, VB.NET	Functiona, load and performance testing
TestLink	Yes	JavaScript, PHP	Test management tool
TestNG	Yes	Java	Test framwork for Java
Testopia	Yes	N/A	Test management tool
TestPlant	No	Java, C#, and Ruby	GUI Automation Testing
TestRail	No	Java, PHP, Python, Ruby, .NET	Test management tool
Testuff	No	N/A	Test management tool
TestWave	No	N/A	Test management tool
Watin	Yes	C#, JavaScript	Web application testing tool
Watir	Yes	Ruby	Web application testing tool
XStudio	No	N/A	Test Management
Zephyr	Yes	N/A	Test Management

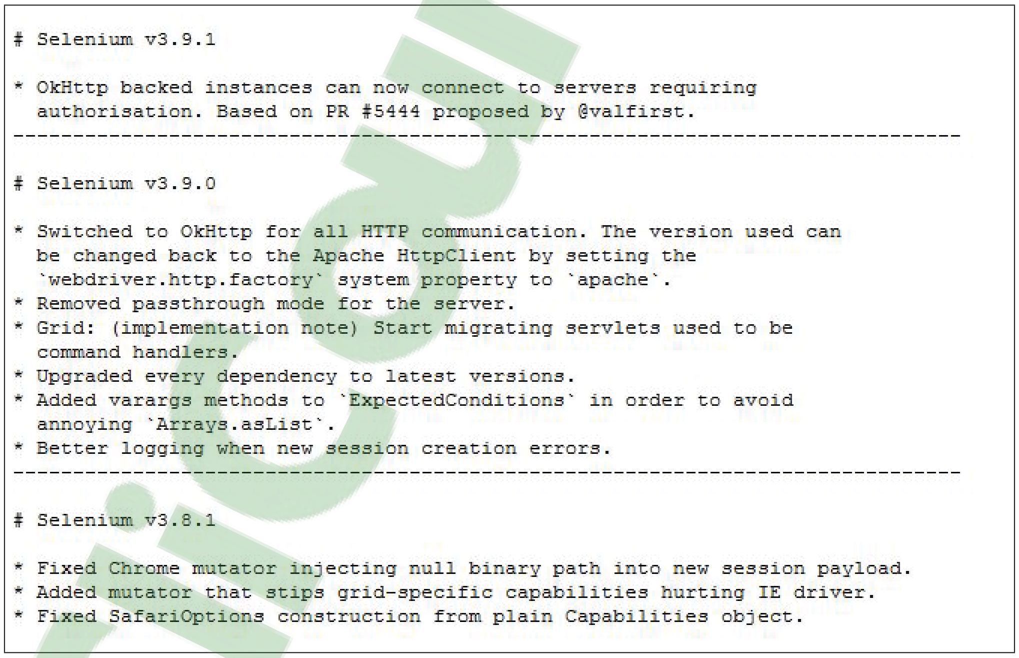
2.2.2 "Trimming" the logs from their noise

change logs and release notes contain various information about the product like new features, bug fixes and etc. (See Figure 2.5). Not all that information is relevant to our study; for instance, a line about a specific bug fix (e.g. a display error) may not be relevant for our study, even if the term *bug* is eminently a testing term. We got rid of the noise (in this context, information not related to a testing activity facilitated or carried out by the tool) by going through the change logs and release notes, sentence by sentence, and removing content not related to testing (See Figure 2.6). This manual trimming process gave us *trimmed logs*(L2) as shown in Figure 2.1.



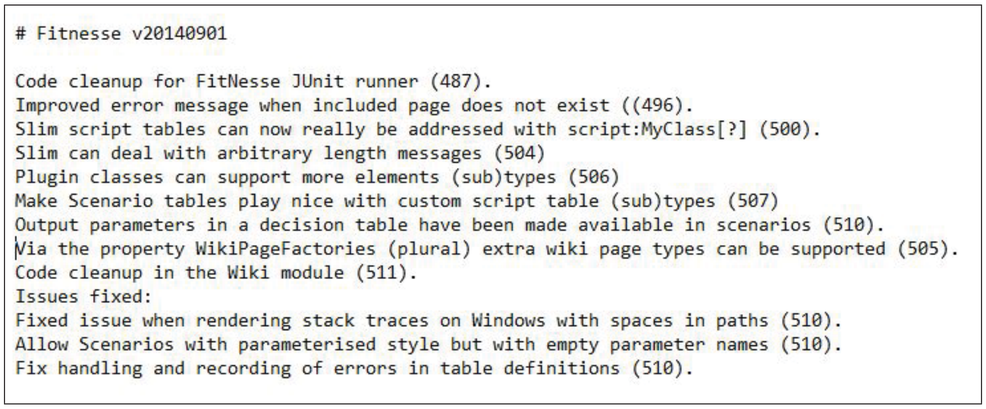
```
v3.9.1
======

* OkHttp backed instances can now connect to servers requiring
  authorisation. Based on PR #5444 proposed by @valfirst.

v3.9.0
======

* Switched to OkHttp for all HTTP communication. The version used can
  be changed back to the Apache HttpClient by setting the
  `webdriver.http.factory` system property to `apache`.
* Removed passthrough mode for the server.
* Grid: (implementation note) Start migrating servlets used to be
  command handlers.
* Upgraded every dependency to latest versions.
* Added varargs methods to `ExpectedConditions` in order to avoid
  annoying `Arrays.asList`.
* Better logging when new session creation errors.

v3.8.1
======

* Fixed Chrome mutator injecting null binary path into new session payload.
* Added mutator that stips grid-specific capabilities hurting IE driver.
* Fixed SafariOptions construction from plain Capabilities object.
```

Figure 2.3　Logs format before uniform



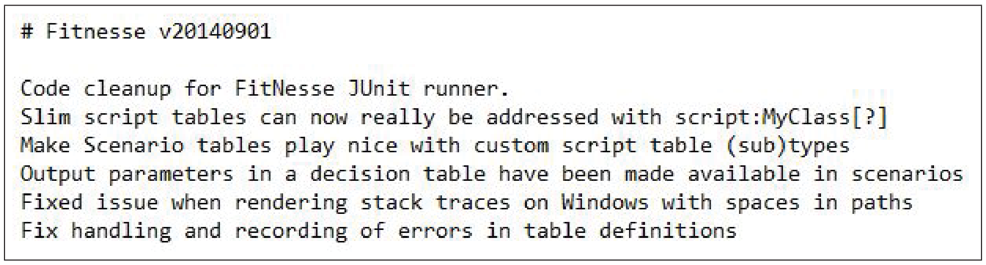
```
# Selenium v3.9.1

* OkHttp backed instances can now connect to servers requiring
  authorisation. Based on PR #5444 proposed by @valfirst.
--------------------------------------------------------------------------------

# Selenium v3.9.0

* Switched to OkHttp for all HTTP communication. The version used can
  be changed back to the Apache HttpClient by setting the
  `webdriver.http.factory` system property to `apache`.
* Removed passthrough mode for the server.
* Grid: (implementation note) Start migrating servlets used to be
  command handlers.
* Upgraded every dependency to latest versions.
* Added varargs methods to `ExpectedConditions` in order to avoid
  annoying `Arrays.asList`.
* Better logging when new session creation errors.
--------------------------------------------------------------------------------

# Selenium v3.8.1

* Fixed Chrome mutator injecting null binary path into new session payload.
* Added mutator that stips grid-specific capabilities hurting IE driver.
* Fixed SafariOptions construction from plain Capabilities object.
```

Figure 2.4　Logs format after uniform



```
# Fitnesse v20140901

Code cleanup for FitNesse JUnit runner (487).
Improved error message when included page does not exist ((496).
Slim script tables can now really be addressed with script:MyClass[?] (500).
Slim can deal with arbitrary length messages (504)
Plugin classes can support more elements (sub)types (506)
Make Scenario tables play nice with custom script table (sub)types (507)
Output parameters in a decision table have been made available in scenarios (510).
Via the property WikiPageFactories (plural) extra wiki page types can be supported (505).
Code cleanup in the Wiki module (511).
Issues fixed:
Fixed issue when rendering stack traces on Windows with spaces in paths (510).
Allow Scenarios with parameterised style but with empty parameter names (510).
Fix handling and recording of errors in table definitions (510).
```

Figure 2.5 Raw logs



```
# Fitnesse v20140901

Code cleanup for FitNesse JUnit runner.
Slim script tables can now really be addressed with script:MyClass[?]
Make Scenario tables play nice with custom script table (sub)types
Output parameters in a decision table have been made available in scenarios
Fixed issue when rendering stack traces on Windows with spaces in paths
Fix handling and recording of errors in table definitions
```

Figure 2.6 Trimmed logs

2.3 Answering our research questions

To answer our research questions, we have to rely extensively on term occurrences in the testing tools' logs. A common and standard tool helpful for this kind of a task is the Porter Stemming algorithm Porter (2001), which is used to stem English words for Information Retrieval (IR) purposes. In short, it helps to unify terms with different spellings but the same meaning, such as *run, ran, runs, running*. In our study, both the terms coming from the established resources and the terms found in the logs were subjected to stemming[4] before trying to retrieve their number of occurrences.

RQ1: To which extent are the terms from established testing's learning resources (SWEBOK v3, ISTQB v3.1) present in change logs and release notes? To answer RQ1, we analyze the frequency of the terms taken from SWEBOK (T1) or ISTQB (T2) in the logs of

[4] Coded a simple script based on the Porter stemmer to retrieve the term occurrences.

the testing tools, whether raw (L1) or trimmed (L2). In particular, for each (Ti, Lj) – with i = 1, 2 and j = 1, 2 – we apply the following steps. We first conduct a quantitative analysis focused on the occurrences of terms from a terminology Ti in the logs Lj. Number of occurrences are examined according to three perspectives: 1) the total number of occurrences of the term in all the logs (TH = *total hits*), 2) the percentage of version releases in which the term appears (VP = *Versions Percentage*), and 3) the percentage of tools in which the term appears (TP = *Tool Percentage*). After such quantitative analyses, we proceed to a qualitative analysis, discussing some of the most significant or frequent terms.

RQ2: Which terms are the most frequent in change logs and release notes?

Focusing on the trimmed logs, we extract a terminology T3 representative of documentation release for software testing. Our research question is then dedicated to the analysis of T3 in a search for the most relevant terms in the testing industry, along different categories (from generic testing terms to technologies, programming languages and tools). Here, we follow the same procedure as in RQ1: first presenting a quantitative analysis with the same measures and then proceeding to a qualitative analysis focused on term occurrences by category.

RQ3: What insights can we get by comparing ISTQB, SWEBOK and our new tool terminology?

In order to answer the third research question of our empirical study, we compared the terminologies T1, T2, T3 in order to understand how much of the tool terminology T3 is present in the terminologies T1 and T2 from the established learning resources. In particular, we take interest in analysing terms that appear in all three terminologies.

CHAPTER 3

RESULTS AND DISCUSSION

In this chapter, we present the results obtained in our empirical study. For all three RQs, we start with a quantitative analysis based on term frequencies in the release documentation of testing tools before proceeding with a qualitative analysis that highlights some interesting terms. The chapter concludes with the presentation of limits and threats to the validity of our study.

3.1 RQ1: To which extent are the terms from established testing's learning resources present in change logs and release notes of testing tools?

Our first research question is basically about projecting the terminology from learning resources to the logs of testing tools and finding out how many of these terms are mentioned (and how often) in the testing tools.

3.1.1 Quantitative results: distribution of terms from the learning resources in the logs

We collected occurrences of terms from a given terminology (SWEBOK or ISTQB) in the release documentation (raw or trimmed), using as axes of analysis, the total number of occurrences of a term (Figure 3.1), the percentage of versions in which a term appears (Figure 3.2), and the percentage of tools in which a term appears (Figure 3.3). Note that the number of terms is *142* for SWEBOK and *725* for ISTQB.

Figure 3.1 displays the distribution of terms from the learning resources, according to their total number of occurrences (H: *hits*) in the release documentation of the selected testing tools. To facilitate the analysis, we classified these occurrences into categories *H0, H1-10, H11-100, H101-1000 and H1001+*; with i) H0: the set of terms that do not appear at all in any release documentation, ii) H1-10: the set of terms that appear between 1 and 10 times, iii) H11-100: the set of terms that appear between 11 and 100 times, iv) H101-1000: the set of terms that

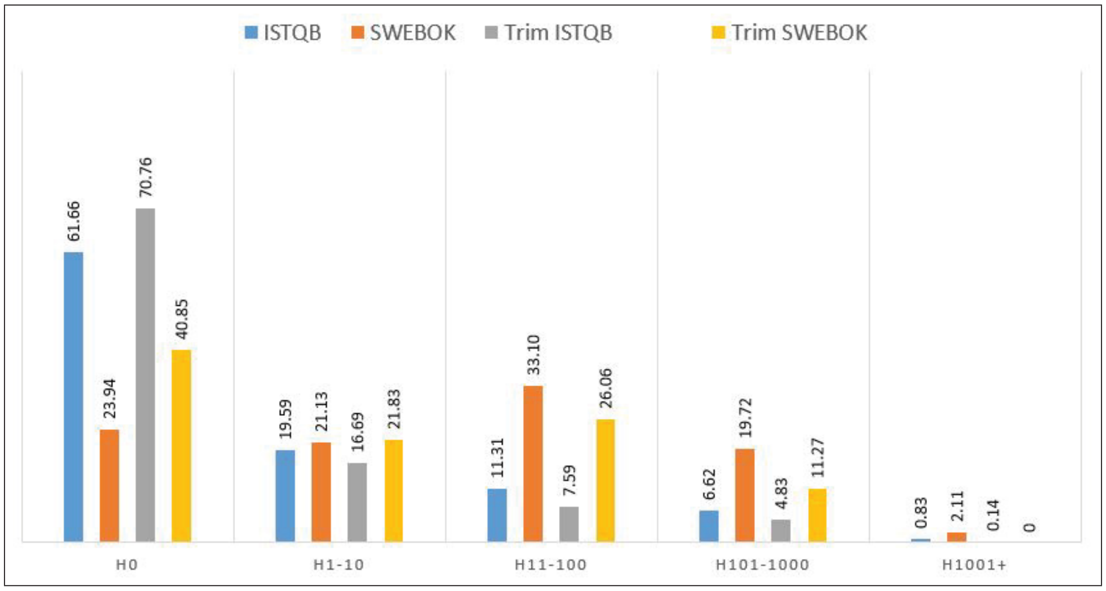


Figure 3.1 Distribution of learning resources' terms relatively to their total hits

appear between 101 and 1000, and v) H1001+: the set of terms that appear more than 1000 times. The first important observation is a binary one and refers to whether a term appears or not in the release documentation. Looking at H0, we can see that 62% of ISTQB terms and 24% of SWEBOK terms do not appear in the raw logs. Unsurprisingly, the percentages are even higher for the trimmed (i.e., less noisy) versions of the logs: 71% of the ISTQB terms and 41% of the SWEBOK terms have zero occurrences.

Additional observations relate to the distribution of terms when they do appear in the logs. Roughly 20% of the terms from either SWEBOK or ISTQB belong to H1-10, meaning these terms do appear but at most 10 times. Differences between SWEBOK and ISTQB become more noticeable for H11-100 (terms that appear between 11 to 100), where SWEBOK terms have percentages about 3 times higher than ISTQB's: 33% vs 11% for raw logs, 26% vs 8% for trimmed logs. The pattern holds for H101-1000, only with percentages about half what they are in H11-100. As for the H1001+ category, the percentage of terms that qualify for that bracket are really low (at most 2%), for all combinations of terminologies and logs. In general, for the H1+ categories, SWEBOK terms present higher percentages than ISTQB's, (except for H1001+ in trimmed logs, for which there are no SWEBOK terms). However, it should be noted that this is not the case for absolute numbers (as opposed to percentages), as there are actually around 5 times more ISTQB terms than SWEBOK's (742 vs 125).

Next, in our analysis, is the distribution of the terms according to their presence in the 1332 release versions of the tools in our study. Figure 3.2 presents the distribution of the terms in the following categories[1]: i) P0: the set of terms that do not appear in any version, ii) P1-5: the set of terms that do appear, but in at most 5% of the versions, iii) P5-10: the set of terms that appear in]5, 10]% of the versions, iv) P10-15: the set of terms that appear in]10, 15]% of the versions, v) P15-20: terms that appear in]15, 20]% of the versions%, and vi) P20-100, the set of terms that appear in more than 20% of the versions.

The observations made for H0 above hold for P0, as a term with 0 occurrences obviously does not appear in any documentation. For P1-5, we notice that i) SWEBOK terms present percentages roughly twice higher than their ISTQB counterparts: 58% vs 32% for raw logs, 49% vs 26% for trimmed logs, and ii) a few – 6 to 9 – percentage points are lost from the raw logs to the trimmed ones. Starting from terms that appear in at least 5%, the percentages for all combinations are dramatically lower, but with SWEBOK terms generally presenting consistently higher percentages than ISTQB's[2].

Finally, we analyse the distribution of the terms according to their presence in the 32 tools in our study (Figure 3.3). After preliminary analysis of the data, we split the distribution into the following categories: i) T0: the set of terms that do not appear in any tool, ii) T1-25: the set of terms that do appear, but in at most 25% of the tools, iii) T26-50: the set of terms that appear in between 26 to 50%, iv) T51-75: the set of terms that appear in between 51 to 75%, and v) T76-100: terms that appear in at least 76% of the tools.

We observe a distribution profile somewhat similar to that of Figure 3.2 (presence in version documentation), with i) most of the terms appearing in between 1% and 25% of the tools, and ii) SWEBOK terms having higher percentages than ISTQB terms. A notable difference here is

[1] We proposed these categories after preliminary analysis of the obtained data.

[2] Again, this is not surprising since there are 5 times more ISTQB terms.

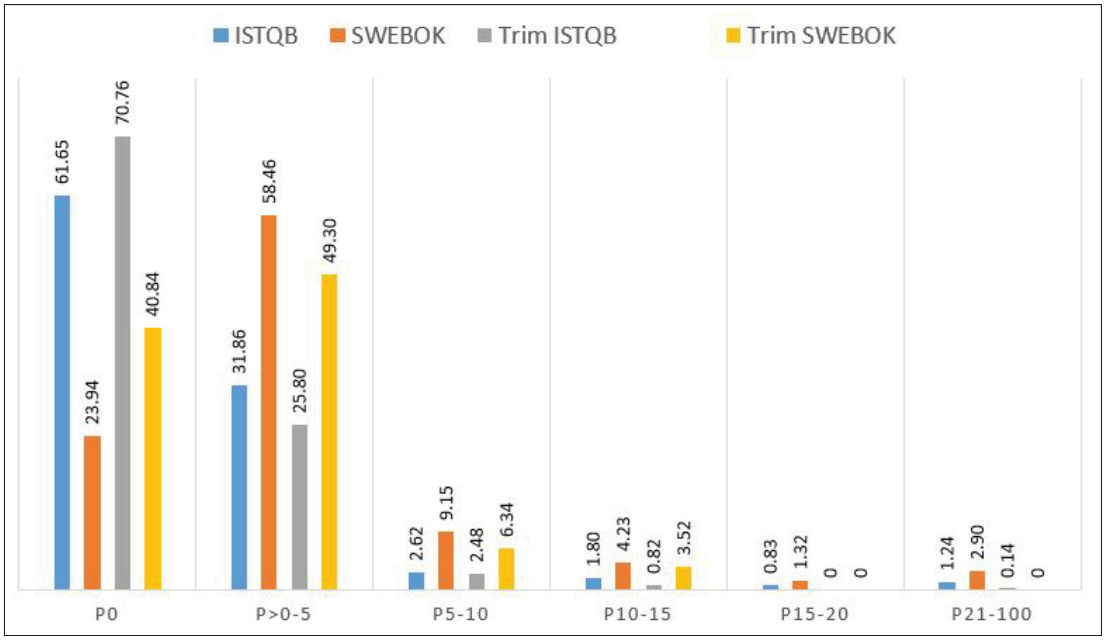


Figure 3.2 Distribution of learning resources' terms relatively to the percentage of versions they appear in

that we can find much higher percentages of terms in higher categories; meaning many terms are indeed mentioned in a large number of tools, even if they are not necessarily mentioned in most versions.

3.1.2 Qualitative analysis: most frequent terms from the learning resources

In this section, we present the most frequent terms from established learning resources in both raw and trimmed release documentation. For each combination of learning resource (SWEBOK or ISTQB) and logs (raw or trimmed), we first retrieve the top 20 most frequent terms, according to the total number of occurrences (H), percentage of mentions in the release versions (P) or mentions in any version of a tool (T). We analyse these terms through their profile respectively to H, P and T (e.g., some terms may be omnipresent in all tools but with relatively few mentions, etc.) and highlight interesting cases. After which, we conduct additional analysis on the whole set of terms (not just the top 20) to identify terms or cases worth discussing. Even though we present results for raw logs, we dedicated more effort to the analysis

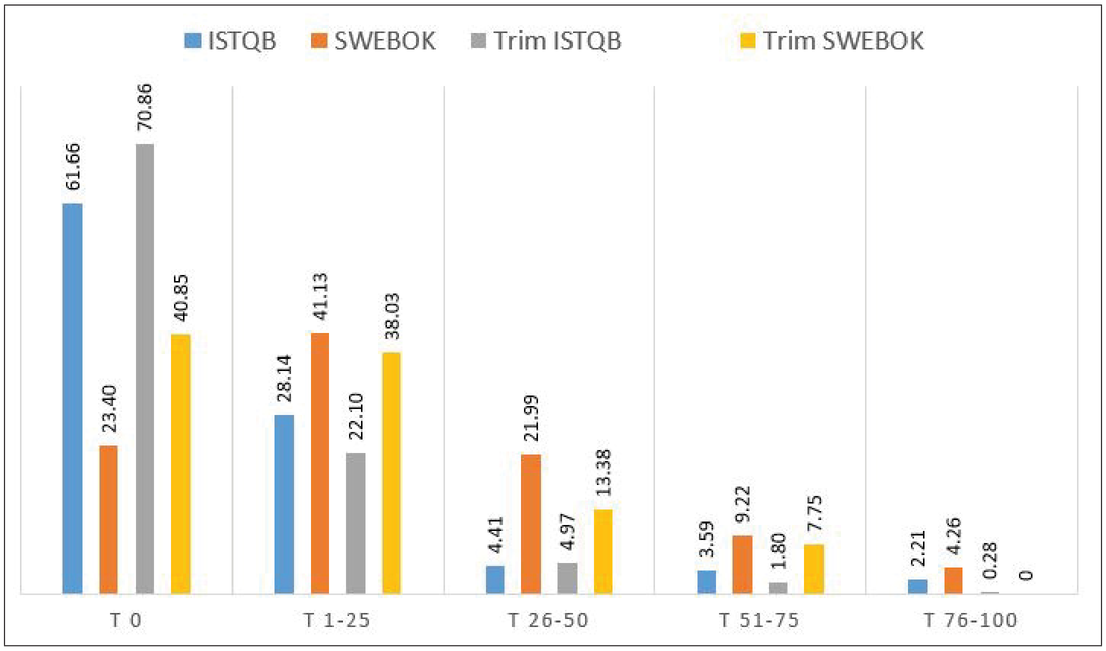


Figure 3.3 Distribution of learning resources' terms relatively to the percentage of tools they appear in

of trimmed logs and thus consequently provide more discussion about them. Note finally that, to ease the reading, we put in italic terms worth highlighting.

3.1.2.1 Most frequent terms from ISTQB

Table 3.1 presents the top 20 most frequent terms from ISTQB in raw logs. The most frequent terms (and consistently so, across number of occurrences, percentages of versions and tools) are terms such as *test, bug, error, result, unit, fail, code*. These are very important, if generic, testing terms. They highlight the focus of most tools in helping finding defects, bugs and errors as well as improving support to fix these bugs through updates and features aimed at improving the test results and logs. Other terms such as *driver, record, function, process* have high number of occurrences (TH) but relatively low presence when it comes to versions and tools; this is unsurprising for a term like *driver*, which would be relevant only for white (or grey) box testing tools. Some other terms such as *test case, project, configur* have high numbers for occurrences (TH) and versions (VA) but are present in only a few tools.

Table 3.1 Frequent term occurrences from ISTQB glossary in raw release documentation.

Terms	Hits	Terms	Versions	Terms	Tools
test	7097	test	57.81	bug	96.88
bug	4021	bug	57.21	test	93.75
driver	1302	error	26.43	error	87.50
result	1067	featur	24.85	result	87.50
error	947	fail	23.05	requir	87.50
unit	846	specif	21.40	featur	84.38
record	826	result	20.57	fail	84.38
fail	819	fault	19.82	fault	84.38
requir	781	requir	19.22	perform	84.38
specif	738	path	18.84	avail	84.38
test case	670	code	17.27	code	81.25
code	640	configur	15.54	oper	78.13
fault	607	function	15.09	path	78.13
function	584	project	14.94	pass	78.13
path	581	integr	14.86	system	78.13
project	578	valid	14.79	specif	75
featur	574	perform	13.51	function	75
configur	413	test case	12.91	unit	75
process	388	record	12.61	valid	68.75
valid	366	unit	12.46	replac	68.75

Table 3.2 presents the most frequent terms from ISTQB glossary in trimmed logs. There is significant similitude with Table 3.1 (raw logs); roughly 80% of the terms in Table 3.2 are also in Table 3.1[3], though with lower numbers as noise was removed from the raw logs. Relatively to the raw logs, there are new terms in the top 20 such as *test plan, test approach, metric, test run, coverage* which have high occurrence (H) in trimmed logs. Our analysis of Table 3.1 and complementary data coming from the logs reveal a few key points worth highlighting:

- Some ISTQB terms are consistently at the top for each of our measures: *bug, test case, error, result, record, fail, unit, function, specif, requir, fault, path, project, integr, valid*. For instance, *record and replay* is an important feature for half of the tools, which allow the quick and easy creation of test cases through the *recording* of all kinds of user-induced

[3] Discussion of those terms in the previous paragraph largely apply here too.

Table 3.2 Most frequent term occurrences from ISTQB glossary in trimmed release documentation

Terms	Hits	Terms	Versions	Terms	Tools
bug	2340	bug	25.23	bug	81.25
test case	579	error	11.49	result	78.13
record	553	test case	11.41	fail	75
result	540	result	11.11	oper	71.88
unit	467	fail	10.44	error	65.63
error	396	unit	10.06	function	65.63
function	354	record	9.23	test case	62.5
fail	341	function	9.16	unit	62.5
specif	336	integr	8.93	path	59.38
test plan	327	path	8.86	valid	59.38
fault	292	specif	8.33	integr	56.25
path	240	requir	7.81	perform	56.25
requir	237	code	7.43	featur	56.25
code	237	fault	7.06	fault	53.13
project	229	valid	6.98	project	50
test approach	214	project	6.31	configur	50
metric	212	configur	5.86	test run	50
integr	185	test run	5.86	failur	50
featur	182	perform	5.63	record	46.88
valid	169	coverag	5.41	specif, requir	46.88

events that can subsequently be replayed across a wide range of desktop, web and mobile applications.

- Others, like *perform, configur, test run*, have modest numbers of hits but relatively high presence in versions and tools; For instance, half the tools place emphasis on and continuously improve their *configur(ability)*, allowing various customisations such as the setting of a time out for test cases, etc.

- Another category of terms, such as *test plan, test approach, metric*, have numbers of hits that are significantly more notable than their presence in versions and tools. The term *metric* is among the top most frequent terms but actually appears in only 5% of the versions and 7% of the tools (i.e. 2 tools);

- A few terms are top ranked for hits and another metric: versions (*code*), or tools (*featur*); in a number of tools, *feature* actually directly refers to the implementation of feature requests from the tool user such as in "feature request: printed test plan includes title of test plan (fman) – resolved"

- Finally, *coverage* is a term mentioned in a lot of versions but with relatively weaker numbers for occurences and tools; it relates to code coverage, support for coverage thresholds, coverage for node tests, test case coverage etc.

3.1.2.2 Most frequent terms from SWEBOK

Table 3.3 presents the top 20 most frequent terms from SWEBOK in raw logs for each metric. The most frequent terms (and consistently so, across number of occurrences, percentages of versions and tools) are terms such as *error, fault, path, test case, valid[ation], integr[ation], failure, API, autom[ation], node, trace, captur[e]*. Other terms such as *interfac, verif, exhaust test* have high number of occurrences but relatively low presence when it comes to versions and tools. Some other terms such as *driver, script, record, defect, coverage* have high numbers for occurrences (Hits) and versions but are present in only a few tools. Finally, some terms are very frequent in tools (*secur, unit test, statist, finit*) and sometimes versions too (*execut, consist, dynam*) but have relatively low numbers of hits.

Table 3.4 presents, relatively to the trimmed logs, the top 20 terms from SWEBOK for either hits, versions or tools. Top 20 terms from these three perspectives amount to 26; 15 terms are present in all the categories: *execut, script, test case, record, error, autom, path, fault, valid, integr, failur, coverag, node, captur, dynam*. Four others are top-ranked in at least 2 categories: hits and versions (*defect, repositori*), versions and tools (*memori leak*), hits and tools (*secur*). Finally, some terms are outstanding according to only one criterion: hits (*interfac, verif*), versions (*consist, driver*), tools (*unit test, finit, trace*).

Table 3.3 Most frequent terms from SWEBOK v3 in raw release documentation

Terms	Total Hits	Terms	Versions	Terms	Tools Appearance
driver	1302	error	26.43	error	87.5
script	958	script	21.92	fault	84.38
error	947	execut	21.7	path	78.13
record	826	fault	19.82	execut	71.88
test case	670	path	18.84	valid	68.75
fault	607	integr	14.86	integr	65.63
path	581	valid	14.79	consist	62.5
valid	366	test case	12.91	test case	59.38
interfac	344	record	12.61	failur	56.25
API	289	API	10.51	record	53.13
defect	278	autom	9.83	API	53.13
integr	261	failur	9.53	autom	53.13
failur	213	driver	9.31	node	50
node	211	node	9.01	secur	50
autom	204	defect	8.78	unit test	46.88
coverag	168	coverag	6.91	trace	43.75
verif	140	consist	5.86	captur	40.63
trace	121	trace	5.26	dynam	40.63
exhaust test	114	captur	4.95	statist	40.63
captur	110	dynam	4.65	finit	40.63

3.1.3 A deeper look at the terms from the learning resources

In this section, we discuss the most significant terms from established source terminologies in trimmed release documentation. We first provide some qualitative insights into the most frequent terms, then extend our scope to 20 more terms that are worth discussing despite not being in the top 20 most frequent.

3.1.3.1 Most frequent terms from ISTQB and SWEBOK

Both ISTQB and SWEBOK have only 26 terms accounting for the top 20 terms for hits, versions and tools. Seven of those terms *test case, record, error, path, fault, valid, integr* are shared between the two resources; they also happen to be in top terms for hits, versions and tools.

Table 3.4 Most frequent terms from SWEBOK v3
in trimmed release documentation

Keywords	**Hits**	**Keywords**	**Versions**	**Keywords**	**Tools**
execut	602	execut	14.56	script	71.88
test case	579	test case	11.41	execut	65.63
record	553	error	11.49	error	65.63
error	396	script	10.89	test case	62.5
script	375	autom	9.76	path	59.38
fault	292	record	9.23	valid	59.38
path	240	integr	8.93	autom	56.25
autom	188	path	8.86	integr	56.25
integr	185	valid	6.98	fault	53.13
valid	169	fault	7.06	failur	50
coverag	132	coverag	5.41	record	46.88
interfac	126	failur	4.8	unit test	43.75
defect	112	defect	4.43	node	37.5
failur	89	node	3.83	captur	37.5
node	83	captur	2.93	memori leak	37.5
secur	65	repositori	2.93	dynam	34.38
captur	62	memori leak	2.48	finit	34.38
verif	62	consist	2.4	coverag	31.25
dynam	55	dynam	2.33	secur	31.25
repositori	53	driver	2.18	trace	31.25

Surprisingly, the term *integr* was not used mainly as in "integration testing" but mostly to indicate that the testing tool could be *integrated* with other tools. Many testing tools such as TestLink, TestNG, Testuff, QMetry etc. outsource advanced issue tracking and test management capabilities to other software such as Jira, Bugzilla, Mantis, TFS, FogBugz. Continuous integration is another major driver for potential "integrations" with tools like Jenkins (from LoadImpact, Ranorex, etc.), Maven (from SoapUI, Jest, MochaJS, JUnit etc.), Tomcat (from Watin and MochaJS), and Gradle (from XStudio, Mockito and JUnit).

3.1.3.2 Significant terms from ISTQB

In this section, we present (see Table 3.5) and discuss some interesting terms from the ISTQB glossary. Based on these terms occurrences and complementary investigations in the release documentation, we make the following observations:

- There are no mentions of Software Development Life Cycles (SDLCs) such as *V-model, Agile and SCRUM*;
- *Memory leaks* appear to be a major issue for a third of the tools, especially when executing scripts for long running record and replay tests;
- *Security* is a concern for about a quarter of the tools; for instance, Soap UI support the Web Services *Security* (WS-Security) extension to apply *security* to web services while other tools support *security* scans on applications;
- *Functional, integration and regression testing* are explicitly mentioned by only one tool out of 8 (12.5%) whereas *load testing* is mentioned and supported by 28% of the tools. Meanwhile, *stress testing* is least represented (only 6.25% of the tools). Finally, only one tool (Test Studio) provides support for *exploratory testing*.

3.1.3.3 Significant terms from SWEBOK

Table 3.6 presents some of the most interesting terms from SWEBOK v3 with respect to their occurrence in trimmed release documentation. Here are some observations:

- Terms referring to *Adhoc, exhaust[ive] test[ing], finit[e] state, test heurist[ics], mutat[ion] and test harness* do not have any occurrences in release documentation. This is not surprising since techniques like Adhoc testing are by definition performed without planning and documentation. As for exhaustive testing, it is not practically possible.
- *Decision tables* are supported only by the tool Fitnesse.

Table 3.5 Significant or frequent terms from ISTQB glossary

Terms	Total Hits	Versions	Appearance in tools
agil test	0	0	0
path coverag	0	0	0
SCRUM	0	0	0
V-model	0	0	0
stress test	2	0.15	6.25 ‘
exploratori test	3	0.23	9.38
reliabl	3	0.23	6.25
integr test	4	0.3	6.25
testabl	5	0.23	6.25
regress test	7	0.45	12.5
test coverag	10	0.3	12.5
function test	22	0.83	12.5
memori leak	38	2.55	34.38
secur	66	2.18	28.13
defect	112	4.43	18.75
coverag	132	5.41	31.25
perform	138	5.63	56.25
load test	152	3.75	28.13
metric	212	4.73	21.88
fail	341	10.44	75
record	553	9.23	46.88

- About 31% of the tools provide support for *traceability* matrix

3.2 RQ2: Which would be the dominant terms in a terminology extracted from change logs and release notes?

The second question of our empirical study shifts the perspective from the learning resources to that of the release documentation terminology. To do so, we focus on terms prominently featured in the release documentation of the studied tools.

Following the trimming from log files of terms unrelated to testing, there were 343 terms left, spanning from testing methodologies and frameworks, to testing practices and technologies. For better insights into these terms, we consider them as a whole but also look into some interesting categories:

Table 3.6 Significant or frequent terms from SWEBOK glossary

Terms	Total Hits	Versions	Appearance in tools
Ad Hoc	0	0	0
exhaust test	0	0	0
finit state	0	0	0
heurist	0	0	0
mutat	0	0	0
test har	0	0	0
interfac test	1	0.08	3.13
reliabl	3	0.23	6.25
testabl	3	0.15	6.25
integr test	4	0.3	9.38
decis tabl	4	0.3	3.13
perform test	13	0.68	3.13
algorithm	27	0.83	12.5
trace	37	2.1	31.25
pattern	43	1.88	25
driver	45	2.18	18.75
verif	62	2.1	25
secur	65	2.1	31.25
graph	97	2.85	25

- **Programming languages** that are supported by the tool; this is usually relevant only for white or grey box testing;
- **Technologies** that are used by the tool, such as XML etc.
- **Generic Testing** terms similar to the ones found in SWEBOK.
- **Tools** that are referenced in testing tools.

In the remaining of this section, we detail observations made on these terms, starting from the whole set of terms then focusing on the different categories above. In essence, we re-apply the analysis in RQ1, but with additional categories.

3.2.1 Distribution of log terminology

Table 3.7 presents the distribution of log terms relatively to their total number of occurrences (H: *hits*) in the trimmed release documentation of software testing tools. Looking at the whole set of terms, we can see that most of the terms have under 10 hits (56%). And the percentages go dramatically down as the number of hits goes higher. This pattern holds for all the subcategories, except for the programming languages, which have relatively high numbers of hits. In fact, these observations apply as well for the percentage of versions (see Table 3.8) and the appearance in tools (see Table 3.9).

Table 3.7 Distribution of total hits for log based terms

Terms	**H0**	**H1-10**	**H11-100**	**H101-1000**	**H1001+**
ALL	0	56.02	29.84	13.87	0.26
Programming language	0	31.25	43.75	25	0
Technologies	0	63.51	29.73	6.76	0
Generic Testing	0	56.93	27.34	15.36	0.37
Tools	0	40	48	12	0

Table 3.8 Distribution of versions' percentage for log based terms

Terms	**P0**	**P1-5**	**P5-10**	**P10-15**	**P15-20**	**P>21 <=100**
ALL	0	90.84	6.81	2.09	0	0.26
Programming language	0	81.25	12.50	6.25	0	0
Technologies	0	97.30	2.70	0	0	0
Generic Testing	0	88.76	8.24	2.62	0	0.37
Tools	0	100	0	0	0	0

Table 3.9 Distribution of tools percentage for log based terms

Terms	**T 0**	**T (1-25)**	**T (26-50)**	**T (51-75)**	**T (76-100)**
ALL	0	81.68	12.83	4.97	0.52
Programming language	0	68.75	18.75	12.50	0
Technologies	0	90.54	8.11	1.35	0
Generic Testing	0	78.65	14.61	5.99	0.75
Tools	0	96	4	0	0

3.2.2 Most frequent terms from log based terminology

Table 3.10 presents the most frequent terms in the logs.

Table 3.10 Most frequent terms from log based terminology

Terms	Hits	Terms	Versions	Terms	Tools appearance
bug	2340	bug	25.23	bug	81.25
test case	630	execut	14.56	fail	75
execut	602	test case	12.54	log	71.88
record	551	java	12.24	script	71.88
fman	536	error	11.34	execut	65.63
github	517	script	10.89	error	65.63
java	510	log	10.59	function	65.63
log	408	fail	10.44	test case	62.50
error	390	autom	9.76	java	62.50
script	375	function	9.16	path	59.38
function	354	record	9.08	window	59.38
fail	341	integr	8.86	har	59.38
specif	336	path	8.78	valid	59.38
test plan	327	xml	8.33	autom	56.25
fault	292	specif	8.33	integr	56.25
xml	240	track	8.26	featur	56.25
requir	237	window	8.18	perform	56.25
path	235	requir	7.81	fault	53.13
firefox	226	valid	7.28	xml	53.13
chrome	217	firefox	7.13	record	46.88

- Some log terms are consistently at the top for each of our measures: *bug, test case, execut, record, java, log, error, script, function, fail, xml, path*; many of these terms were already prominent in RQ1 but there are new ones such as xml and java.
- Others, like *autom, integr, window, valid*, have modest numbers of hits but relatively high presence in versions and tools;
- Another category of terms, such as *fman, github, test plan, chrome*, have numbers of hits that are significantly more notable than their presence in versions and tools.

- A few terms are top ranked for hits and another metric: versions (*specif, requir, firefox*), or tools (*fault*);
- Finally, terms such as *har, featur, perform* are mentioned in many tools but have relatively weaker numbers for occurrences and versions;

3.2.3 Generic Testing terms in the logs

Table 3.11 present the most frequent generic testing terms. Terms that were not previously highlighted in RQ1 include *project, test run, bug fix, debug, test result* etc.

3.2.4 Technologies mentioned in logs

Distribution of technology-related terms from log based terminology is presented in Table 3.12. Following manual inspections of the logs, we can make the following observations:

- 59.4% of the tools run on *Windows* compared to 12.5% for *Mac OS*;
- 37.5% of the tools generate test data in JavaScript Object Notation (*JSON*) format;
- Unicode Transformation Format (*UTF*-8) character encoding is explicitly mentioned and supported by 34.4% of the tools in our study;
- Relatively to web page testing with a browser, *Firefox* leads with a support by 31.3% of the tools, with *Chrome* a close second (28.1%) and other browsers (*Internet Explorer (IE), Safari and Opera*) being supported by 21.9% of the tools. *PhantomJS* is the least popular browser, and is supported by only 9.4% of tools;
- The web hosting service (*GitHub*) is used by 28.13% of the tools.
- *SOAP* (Simple Object Access Protocol) web services and APIs are mentioned by 21.88% of tools.

.

Table 3.11 Distribution of generic testing terms

Terms	Totalhits	Versions	Tools appearance
function	354	9.16	65.63
test case	630	12.54	62.5
valid	169	7.28	59.38
integr	179	8.86	56.25
featur	173	5.33	56.25
perform	138	5.63	56.25
project	229	6.31	50
test run	108	6.01	50
record	551	9.08	46.88
bug fix	118	6.31	46.88
debug	85	4.73	43.75
test result	48	2.85	43.75
unit test	40	2.1	43.75
variabl	163	3.45	40.63
test suit	194	5.86	37.5
memori leak	37	2.48	34.38
coverag	132	5.41	31.25
secur	65	2.1	31.25
test execut	150	3.98	28.13
test report	28	1.58	28.13
test approach	214	2.03	25
execut test	67	2.33	25
verif	62	2.1	25
standard	27	1.5	25
test plan	327	4.95	21.88
metric	212	4.73	21.88
bug report	51	2.18	21.88
regular express	26	1.2	21.88
test script	50	1.13	18.75
test summari report	40	1.05	18.75

3.2.5 Programming languages from Logs

Table 3.13 presents the distribution of programming languages in log terms. We make the following observations of these terms:

Table 3.12 Distribution of technology terms

Terms	Totalhits	Versions	Tools appearance
window	201	8.18	59.38
json	90	3.15	37.5
node	83	3.83	34.38
utf	36	2.1	34.38
firefox	226	7.13	31.25
github	517	4.05	28.13
chrome	217	4.13	28.13
linux	67	3.15	25
soap	87	3.53	21.88
internet explor	54	2.78	21.88
safari	36	1.95	21.88
opera	13	0.83	21.88
bugtrack	46	2.33	12.5
mac os	31	1.28	12.5
algorithm	27	0.83	12.5
chrome extens	5	0.38	12.5
ie record	69	1.58	9.38
bug tracker	67	3.68	9.38
mysql	26	1.28	9.38
phantomjs	8	0.45	9.38

- *Java* is the leading programming language when it comes to write the test cases or the test scripts; it is backed by 63% of the tools;

- 43.75% of the tools support the testing of .NET applications on Windows platform;

- The query language *XPath* has high numbers of hits and versions but it is supported by only 25% of the tools. The inverse is observed for *JavaScript*, which has relatively low numbers for hits and versions but is supported by 37.5% of the tools;

- *Python* is supported by only 15.63% of the tools for the writing of test cases or test scripts;

- Finally, *Groovy script, IronPython, java swing and XQuery* are the least popular programming languages supported by the testing tools.

Table 3.13 Distribution of programming language terms

Terms	Totalhits	Versions	Tools appearance
java	510	12.24	62.5
net	186	5.93	43.75
javascript	63	2.48	37.5
xpath	92	4.13	25
sql	122	4.88	21.88
python	37	2.03	15.63
php	55	1.8	12.5
visual basic	14	0.9	9.38
perl	4	0.15	6.25
groovi script	3	0.15	6.25
ironpython	2	0.15	3.13
java swing	1	0.08	3.13
xquery	1	0.08	3.13

3.2.6 Tools mentioned in the logs

Table 3.14 presents tools that are mentioned in the release documentation of the testing tools. Many tools are integrated with others to make them standalone and robust. Here are some key observations:

- *Jira* (25% of the tools) and *Bugzilla* (15.63% of the tools) are the most popular issue tracking systems when it comes to tool integration;
- As for build automation tools, 21.88% of tools provides support for *Maven*;
- *Selenium, TestNG and Ranorex* are popular testing frameworks for the testing of desktop or web-based or mobile applications; *Selenium* can be integrated with 18.75% of tools for testing purposes and report generation;
- Only, 6.25% tools provide integration support for tools such as *Sahi, SoapUI, XUnit, XStudio, Fitnesse, and TestPlant*.

Table 3.14 Distribution of the tools mentioned in the logs

Terms	Totalhits	Versions	Tools appearance
jira	129	4.65	25
maven	52	1.58	21.88
selenium	30	1.35	18.75
testng	64	2.48	15.63
bugzilla	41	2.25	15.63
ranorex	128	3.6	6.25
sahi	29	1.5	6.25
soapui	23	1.35	6.25
xunit	15	0.9	6.25
xstudio	12	0.9	6.25
load gener	9	0.38	6.25
fitness	3	0.23	6.25
testplant	3	0.23	6.25
testlink	31	0.9	3.13
testuff	12	0.83	3.13
watin	9	0.6	3.13
test studio	15	0.45	3.13
funkload	11	0.38	3.13
sikulix	4	0.15	3.13
test plant	1	0.08	3.13
testwav	1	0.08	3.13

3.3 RQ3: Comparing ISTQB, SWEBOK and Log based terminologies

In this section, we compare terminologies derived from learning resources and release documentation. We compute intersections of the terminologies two at a time (SWEBOK and logs[4], ISTQB and logs[5]), then all three at once. The set of terms present in all three terminologies is reported in Table 3.15. Detailed lists of the intersections are presented in Appendix I.

The questions of interest here, for each learning resource, is: a) which percentage of its terminology is part of the documentation release terminology, b) which percentage of the documentation release terminology can be found in the learning resource terminology. Considering

[4] Intersection of log based *vs* SWEBOK v3 terms = 80

[5] Intersection of log based *vs* ISTQB glossary = 173

the SWEBOK terminology, 29.96% of it are terms from the documentation release, and those terms account for 56.33% of the documentation release terminology. As for the ISTQB terminology, 23.86% of it are terms from the documentation release, and those terms account for 64.79% of the documentation release terminology. This means that most terms from either learning resource are not used in the documentation release terminology but on a slightly brighter side, the resources terminologies do contain more than half – almost two-thirds for ISTQB – of the documentation release terminology.

Table 3.15 Intersection of all sets terminology

interfac test, reliabl, testabl, accuraci, stress test, test level, decis tabl, matur, test log, test case, error, secur, function test, coverag, robust, integr, path, valid, qualif, accept, integr test, perform test, usabl, consist, fault, oracl, unit test, instrument, stub, failur, record, defect, verif, driver

3.4 Threats to validity

The purpose of this section is to describe the elements that could impact the validity of this study with regard to the external and internal validity.

External validity concerns the generalizability of our results. We cannot claim that our results can be generalized to all testing tools. The tools in our study are those, mostly free, for which we were able to obtain release documentation. Furthermore, we had to exclude the tools for which release documentation were not up to date. Nonetheless, our study unveils some interesting facts that deserve further investigation.

Internal validity threats refer to factors that could impact our conclusions. At all steps of our study, we proceed to rigorous verification of the data: from the extraction of the terminologies to the processing of the terms distributions. A few points of concern would be with respect to the use of stemming, which sometimes leads to semantically different terms getting reduced to the same root. Though we tried to be as thorough as possible in identifying these inaccuracies, we cannot guarantee there are no residual problems there. Another possible problem is that in some release documentation, important information could be missing from the change logs of a version.

CONCLUSION AND RECOMMENDATIONS

In this thesis, we presented an empirical analysis of the terminology of software testing tools through their release documentation. To do so, we selected a sample of software testing tools and focused on their release documentation. Terminology extracted from these documents was used to assess possible discrepancies between those of well established learning resources.

The results obtained showed that there are indeed significant differences between the terminology of learning resources and that of testing tools. Our study also unveiled some insights on concepts that are the most represented in testing tool terminology, from generic testing terms to technologies, programming languages and even other tools.

Future work could involve a trend analysis aimed at retrieving which terms are picking up steam over the last years. This could provide some insights on the tooling support for some testing concepts. For instance, mutation testing is relatively recently being made accessible through new tools. Additionally, and longer term, there could be avenues for a research work directed towards providing a terminology-based approach for the selection of a testing tool based on a list of requirements and desired features.

APPENDIX I

INTERSECTION OF ESTABLISHED REFERENCE TERMS AND LOG BASED TERMS

Intersection of log based terms and SWEBOK v3 terms.

Terms intersection between log based terms and SWEBOK v3 terms are as followed:

Interfac test, reliabl, testabl, accuraci, stress test, track, test level, techniqu, decis tabl, determinist, matur, captur, reus, dynam, loop, test log, limit, target, statist, model, recoveri, white box, interfac, test case, machin, trace, error, estim, test select, graph, structur, secur, function test, coverag, defect track, exploratori, robust, script, random, integr, autom, deriv, execut, defect, speed, repositori, path, valid, increment, qualif, accept, integr test, frequenc, usabl, consist, code base, perform test, fault, unit test, oracl, finit, cognit, environ, log, pattern, protocol, equival, criteria, stub, score, har, effort, failur, instrument, regress, effect, record, profil, verif, driver.

Intersection of log based terms and ISTQB terms.

Terms intersection between log based terms and ISTQB terms are as followed:

Memori leak, specif, pass, recover, decis tabl, test requir, function, test run, test log, compat test, statement, test session, configur manag, test script, featur, test pass, tester, dashboard, secur, mainten, stabil, fail, except handl, scenario test, complex, decis, qualif, domain, test case design techniqu, indic, configur test, usabl, consist, test stage, project, process, precondit, risk, oper, test driver, test suit, record, bug report, driver, prioriti, problem , interfac test, testabl, accuraci, stress test, test level, static test, moder, test infrastructur, test approach, static analysi, test autom, deliver, block test case, test process, debug, bug, data driven test, maintain, review, test compar, convers test, field test, traceabl, script test, test plan, path test, perform test, perform, test data, complet test, input valu, debugg, test environ, qualiti, entri point, regress test, test result, verif, test target, reliabl, test record, test execut phase, expect result, matur, test schedul, metric, accept test, standard, emul, standard test, requir, inspector, test execut autom, test cycl, coverag, integr, test implement, path, valid, concurr test, test manag tool, best practic, test tool, fault, usabl test, unit test, pointer, defect report, test specif, compil, output, safeti, system, stub, failur, configur item, exploratori test, test summari report, test coverag, link test, user scenario test, test object, test basi, test case, dead code, interoper, buffer overflow, defect manag, audit trail, error, variabl, mileston, test manag, input, dynam test, audit, branch, scal-abl, script languag, test fail, function test, storag, test report, robust, suitabl, servic test, condit, coverag analysi, accept, test oracl, test type, integr test, secur test, test design, simul, oracl, instrument, code, test execut, test case specif, certif, configur, code coverag, compon, defect, result, test data prepar tool.

BIBLIOGRAPHY

Surafel Lemma Abebe, Venera Arnaoudova, Paolo Tonella, Giuliano Antoniol, and Yann-Gael Gueheneuc. Can lexicon bad smells improve fault prediction? In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 235–244. IEEE, 2012.

Surafel Lemma Abebe, Nasir Ali, and Ahmed E Hassan. An empirical study of software release notes. *Empirical Software Engineering*, 21(3):1107–1142, 2016.

A Abran, JW Moore, P Bourque, R Dupuis, and LL Tripp. Swebok: Software engineering body of knowledge trial version. *IEEE Computer Society, US*, 2001.

Alain Abran, JW Moore, P Bourque, R Dupuis, and LL Tripp. Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess*, 2004.

Aakash Ahmad, Pooyan Jamshidi, Muteer Arshad, and Claus Pahl. Graph-based implicit knowledge discovery from architecture change logs. In *Proceedings of the WICSA/ECSA 2012 Companion Volume*, pages 116–123. ACM, 2012.

Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. What's a typical commit? a characterization of open source software repositories. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 182–191. IEEE, 2008.

Giuliano Antoniol, Massimiliano Di Penta, and Ettore Merlo. An automatic approach to identify class evolution discontinuities. In *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*, pages 31–40. IEEE, 2004.

Harsh P. Bajaj. Choosing the right automation tool, 2014.

Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, pages 85–103. IEEE Computer Society, 2007.

Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143. ACM, 2006.

P Bourque and RE Fairley. Guide to the software engineering body of knowledge (swebok)2015: Iso - international organization for standardization, Sep 2015. URL https://www.iso.org/standard/33897.html.

Pierre Bourque, Richard E Fairley, et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Tilmann Bruckhaus. *A Quantitative Approach for Analyzing the Impact of Tools on Software Productivity*. PhD thesis, McGill University, Montreal, Que., Canada, Canada, 1997. AAINQ36962.

Jim Buffenbarger and Kirk Gruell. What have you done for me lately?(branches, merges, and change logs). In *International Workshop on Software Configuration Management*, pages 18–24. Springer, 1997.

Raymond PL Buse and Westley R Weimer. Automatically documenting program changes. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 33–42. ACM, 2010.

Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. How changes affect software entropy: an empirical study. *Empirical Software Engineering*, 19(1): 1–38, 2014.

Kai Chen, Stephen R Schach, Liguo Yu, Jeff Offutt, and Gillian Z Heller. Open-source change logs. *Empirical Software Engineering*, 9(3):197–210, 2004.

Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. *IEEE Transactions on software engineering*, 29(3):210–224, 2003.

Seyed Amir Emami, Jason Chin Lung Sim, and Kwan Yong Sim. A survey on open source software testing tools: a preliminary study in 2011. In *Fourth International Conference on Machine Vision (ICMV 11)*, pages 83502Y–83502Y. International Society for Optics and Photonics, 2011.

Anthony Finkelstein and John Dowell. A comedy of errors: the london ambulance service case study. In *Proceedings of the 8th International Workshop on Software Specification and Design*, page 2. IEEE Computer Society, 1996.

Gordon Fraser, Franz Wotawa, and Paul E Ammann. Testing with model checkers: a survey. *Software Testing, Verification and Reliability*, 19(3):215–261, 2009.

Vahid Garousi and Michael Felderer. 1 living in two different worlds: A comparison of industry and academic focus areas in software testing, Dec 2016. URL https://www.researchgate.net/publication/311913424_Living_in_two_different_worlds_A_comparison_of_industry_and_academic_focus_areas_in_software_testing.

Vahid Garousi and Junji Zhi. A survey of software testing practices in canada. *Journal of Systems and Software*, 86(5):1354–1376, 2013.

Vahid Garousi, Matt M Eskandar, and Kadir Herkiloğlu. Industry–academia collaborations in software testing: experience and success stories from canada and turkey. *Software Quality Journal*, pages 1–53, 2016.

D German. Using software trails to rebuild the evolution of software, elisa 2003 workshop, evolution of large-scale industrial software applications, 24 sept., amsterdam, 2004.

Daniel M German. An empirical study of fine-grained software modifications. *Empirical Software Engineering*, 11(3):369–393, 2006.

GNU. Gnu coding standards: Style of change logs. URL https://www.gnu.org/prep/standards/html_node/Style-of-Change-Logs.html#Style-of-Change-Logs.

ISTQB Glossary Working Group et al. Standard glossary of terms used in software testing. Technical report, Technical report, International Software Testing Qualifications Board. Version 3.01, 2015.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, 19(2): 167–199, 2012.

Peter Hutterer. Who-t, Dec 2009. URL http://who-t.blogspot.ca/2009/12/on-commit-messages.html.

IEEE. Ieee guide for software verification and validation plans. *IEEE Std 1059-1993*, pages i–87, 1994. doi: 10.1109/IEEESTD.1994.121430.

ISTQB. Certifying software testers worldwide, 2016. URL http://www.istqb.org/.

Georg Kaes and Stefanie Rinderle-Ma. Mining and querying process change information based on change trees. In *International Conference on Service-Oriented Computing*, pages 269–284. Springer, 2015.

Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice*, 19(2):77–131, 2007.

Chris F. Kemerer and Sandra Slaughter. An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering*, 25(4):493–509, 1999.

Foutse Khomh, Tejinder Dhaliwal, Ying Zou, and Bram Adams. Do faster releases improve software quality?: an empirical case study of mozilla firefox. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 179–188. IEEE Press, 2012.

Miryung Kim, David Notkin, Dan Grossman, and Gary Wilson. Identifying and summarizing systematic code changes via rule inference. *IEEE Transactions on Software Engineering*, 39(1):45–62, 2013.

Edward Kit and Susannah Finzi. *Software Testing in the Real World: Improving the Process*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995. ISBN 0-201-87756-2.

Olivier Lacan. Keep a changelog, 2016. URL http://keepachangelog.com/en/1.0.0/.

Claude Y Laporte, B Nabil, and D Mikel. Measuring the cost of software quality of a large software project at bombardier transportation: a case study. *Software Qual. Manage*, 14 (3):14–31, 2012.

Kaiping Liu, Hee Beng Kuan Tan, and Hongyu Zhang. Has this bug been reported? In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 82–91. IEEE, 2013.

David Lo, Nachiappan Nagappan, and Thomas Zimmermann. How practitioners perceive the relevance of software engineering research. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 415–425. ACM, 2015.

Walid Maalej and Hans-Jörg Happel. Can development work describe itself? In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 191–200. IEEE, 2010.

marketshare. Search engine market share, Oct 2017. URL https://www.netmarketshare.com/search-engine-market-share.aspx?qprid=4&qpcustomd=0.

Paul W McBurney and Collin McMillan. Automatic documentation generation via source code summarization of method context. In *Proceedings of the 22nd International Conference on Program Comprehension*, pages 279–290. ACM, 2014.

James Bret Michael, Bernard J Bossuyt, and Byron B Snyder. Metrics for measuring the effectiveness of software-testing tools. In *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, pages 117–128. IEEE, 2002.

Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K Vijay-Shanker. Automatic generation of natural language summaries for java classes. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 23–32. IEEE, 2013.

Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 484–495. ACM, 2014.

Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. Arena: An approach for the automated generation of release notes. *IEEE Transactions on Software Engineering*, 2016.

Vicky Mosley. How to assess tools efficiently and quantitatively. *IEEE Software*, 9(3):29–32, 1992.

Khaled M Mustafa, Rafa E Al-Qutaish, and Mohammad I Muhairat. Classification of software testing tools based on the software testing methods. In *Computer and Electrical Engineering, 2009. ICCEE'09. Second International Conference on*, volume 1, pages 229–233. IEEE, 2009.

G.J. Myers. *The Art of Software Testing*. A Wiley-Interscience publication. Wiley, 1979a. ISBN 9780471043287. URL https://books.google.ca/books?id=DV0ZAQAAIAAJ.

Glenford J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979b. ISBN 0471043281.

Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley Publishing, 3rd edition, 2011. ISBN 1118031962, 9781118031964.

Myung Hwan Na, Jongwoo Jeon, and Dong Ho Park. Testing whether failure rate changes its trend with unknown change points. *Journal of statistical planning and inference*, 129 (1):317–325, 2005.

Crescencio Rodrigues Lima Neto, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A software product lines system test case tool and its initial evaluation. In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, pages 25–32. IEEE, 2012.

Lima Neto and Crescencio Rodrigues. Splmt-te: A software product lines system test case tool, 2011.

online repository for testing tools. list of testing tools. http://qatestingtools.com/, 2015.

Kai Pan, E James Whitehead, and Guozheng Ge. Textual and behavioral views of function changes. In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 8–13. ACM, 2005.

Martin F Porter. Snowball: A language for stemming algorithms, 2001.

Robert M. Poston and Michael P. Sexton. Evaluating and selecting testing tools. In *Assessment of Quality Software Development Tools, 1992., Proceedings of the Second Symposium on*, pages 55–64. IEEE, 1992.

Tom Preston-Werner. Semantic versioning 2.0. 0. *línea]. Available: http://semver. org*, 2013.

QATestingTools.com. All about software testing tools. accessed 22 october 2015. url http://www.qatestingtools.com/. URL http://www.qatestingtools.com/contact-consultant.

Jane Radatz, Anne Geraci, and Freny Katki. Ieee standard glossary of software engineering terminology. *IEEE Std*, 610121990(121990):3, 1990.

Sarah Rastkar and Gail C. Murphy. Why did this code change? In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1193–1196, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3. URL http://dl.acm.org/citation.cfm?id=2486788.2486959.

Stuart Reid. The art of software testing, second edition. glenford j. myers. revised and updated by tom badgett and todd m. thomas, with corey sandler. john wiley and sons, new jersey, u.s.a., 2004. isbn: 0-471-46912-2, pp 234: Book reviews. *Softw. Test. Verif. Reliab.*, 15 (2):136–137, June 2005. ISSN 0960-0833. doi: 10.1002/stvr.v15:2. URL http://dx.doi.org/10.1002/stvr.v15:2.

SB Rinderle, Martin Jurisch, and Manfred Reichert. On deriving net change information from change logs-the deltalayer-algorithm. 2007.

Jelber Sayyad Shirabad, Timothy C Lethbridge, and Stan Matwin. Supporting software maintenance by mining software update records. In *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, pages 22–31. IEEE, 2001.

speak your languages. Why terminology is important, 2017. URL http://www.speakyourlanguages.com/courses/unit01/ter03/01ter03.htm.

Gregory Tassey. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project*, 7007(011), 2002.

Sergey Uspenskiy et al. A survey and classification of software testing tools. 2010.

Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing. 2006.

Ming-Shi Wang and Jung-te Weng. Locating matching rules by mining software change log. In *9th Joint International Conference on Information Sciences (JCIS-06)*. Atlantis Press, 2006.

Tom Preston Werner. Semantic versioning 2.0.0, 2016. URL http://semver.org/.

Wikipedia. Category:software testing tools, Nov 2015. URL https://en.wikipedia.org/wiki/Category:Software_testing_tools.

Pulei Xiong. *Life-cycle e-commerce testing with object-oriented TTCN-3*. PhD thesis, University of Ottawa (Canada), 2004.

Christine Youngblut. An examination of selected software testing tools: 1992. Technical report, DTIC Document, 1992.

Liguo Yu. Mining change logs and release notes to understand software maintenance and evolution. *CLEI Electron Journal*, 12(2):1–10, 2009.