Information and Software Technology 58 (2015) 123–147

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Software test-code engineering: A systematic mapping

Vahid Garousi Yusifoğlu [a,b,c,*], Yasaman Amannejad [b], Aysu Betin Can [d]

[a] System and Software Quality Engineering Research Group (SySoQual), Department of Software Engineering, Atilim University, Kızılcaşar Mahallesi, İncek, Ankara, Turkey
[b] Software Quality Engineering Research Group (SoftQual), Department of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta, Canada
[c] Maral Software Consulting Corporation A.Ş., Ankara, Turkey
[d] Software Technologies Research Group, Informatics Institute, Middle East Technical University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 30 May 2013
Received in revised form 6 June 2014
Accepted 10 June 2014
Available online 2 July 2014

Keywords:
Systematic mapping
Survey
Study repository
Software test-code engineering
Development of test code
Quality assessment of test code

ABSTRACT

Context: As a result of automated software testing, large amounts of software test code (script) are usually developed by software teams. Automated test scripts provide many benefits, such as repeatable, predictable, and efficient test executions. However, just like any software development activity, development of test scripts is tedious and error prone. We refer, in this study, to all activities that should be conducted during the entire lifecycle of test-code as Software Test-Code Engineering (STCE).
Objective: As the STCE research area has matured and the number of related studies has increased, it is important to systematically categorize the current state-of-the-art and to provide an overview of the trends in this field. Such summarized and categorized results provide many benefits to the broader community. For example, they are valuable resources for new researchers (e.g., PhD students) aiming to conduct additional secondary studies.
Method: In this work, we systematically classify the body of knowledge related to STCE through a systematic mapping (SM) study. As part of this study, we pose a set of research questions, define selection and exclusion criteria, and systematically develop and refine a systematic map.
Results: Our study pool includes a set of 60 studies published in the area of STCE between 1999 and 2012. Our mapping data is available through an online publicly-accessible repository. We derive the trends for various aspects of STCE. Among our results are the following: (1) There is an acceptable mix of papers with respect to different contribution facets in the field of STCE and the top two leading facets are tool (68%) and method (65%). The studies that presented new processes, however, had a low rate (3%), which denotes the need for more process-related studies in this area. (2) Results of investigation about research facet of studies and comparing our result to other SM studies shows that, similar to other fields in software engineering, STCE is moving towards more rigorous validation approaches. (3) A good mixture of STCE activities has been presented in the primary studies. Among them, the two leading activities are quality assessment and co-maintenance of test-code with production code. The highest growth rate for co-maintenance activities in recent years shows the importance and challenges involved in this activity. (4) There are two main categories of quality assessment activity: detection of test smells and oracle assertion adequacy. (5) JUnit is the leading test framework which has been used in about 50% of the studies. (6) There is a good mixture of SUT types used in the studies: academic experimental systems (or simple code examples), real open-source and commercial systems. (7) Among 41 tools that are proposed for STCE, less than half of the tools (45%) were available for download. It is good to have this percentile of tools to be available, although not perfect, since the availability of tools can lead to higher impact on research community and industry.
Conclusion: We discuss the emerging trends in STCE, and discuss the implications for researchers and practitioners in this area. The results of our systematic mapping can help researchers to obtain an overview of existing STCE approaches and spot areas in the field that require more attention from the research community.

* Corresponding author at: System and Software Quality Engineering Research Group (SySoQual), Department of Software Engineering, Atilim University, Kızılcaşar Mahallesi, İncek, Ankara, Turkey.
E-mail addresses: vahid.garousi@atilim.edu.tr (V. Garousi Yusifoğlu), yasaman.amannejad@ucalgary.ca (Y. Amannejad), betincan@metu.edu.tr (A. Betin Can).

http://dx.doi.org/10.1016/j.infsof.2014.06.009
0950-5849/

Contents

1. Introduction

Automated software testing and development of test code (scripts) are now mainstream in the software industry. For instance, in a recent book, Microsoft test engineers reported that "*there were more than a million [automated] test cases written for Microsoft Office 2007*" [37]. As another example, for the version 2.1 of the Android smart-phone software platform, there were about 2.1 million lines of code (LOC), from which about 358,000 LOC were Java test code written in the JUnit framework [61,62].

Also, according to a survey in year 2009 across the Canadian province of Alberta which was conducted to understand and characterize the industrial software testing practices [63], it is stated that about 65% and 45% of companies in Alberta automate their unit and system testing tasks, respectively.

Automated test scripts provide many benefits, such as repeatable, predictable, and efficient test executions. However,

development of test-code scripts is tedious, error prone and requires significant up-front investment [50]. Moreover, after the initial development, like any software engineering artifact, test code requires quality assessment and maintenance. Thus, test scripts have a lifecycle involving different development, quality assessment (verification and validation), and maintenance activities, and must co-evolve with the application code under test. To perform such activities efficiently (especially when the test suite of an application grows over time to consist of tens of thousands of scripts), the development of appropriate techniques, tools, and methods that encompass the entire test-script lifecycle is essential. We refer to all activities that should be conducted during the entire lifecycle of test-code as Software Test-Code Engineering (STCE).

To address the above challenges associated with STCE, many researchers from academia (e.g., [4]) and also practitioners (e.g., [23]) have tackled the STCE challenges and have proposed various methods, techniques, tools, and metrics in this area. According to an initial literature search that we conducted, as of this writing (Spring 2013), about 72 studies (studies and books) have appeared since 1999 in the area of STCE. As this research area matures and the number of related studies increases, it is important to systematically classify the current state-of-the-art and to provide an overview of the trends in this specialized field [64–66]. Such categorized results provide many benefits to the broader community of researchers and practitioners. For example, they would be valuable resources for new researchers (e.g., PhD students) aiming to identify open research areas needing further work and/or to conduct additional *secondary studies* [64–66]. A secondary study in this context is defined as a study of (primary) studies [67].

In this work, we systematically analyze and classify the body of knowledge related to STCE through a systematic mapping (SM) study [66]. As part of this study, we pose a list of research questions, define inclusion (selection) and exclusion criteria of relevant studies, and systematically develop and refine a systematic map (classification schema) of all the selected studies.

After a careful selection process (Section 4), our study pool includes a set of 60 studies (from the set of 72 identified studies) published in the area of STCE between 1999 and 2012. The full version of our mapping data is available through a publicly-accessible online repository [68]. We derive the trends for instance in terms of types of studies (types of contribution and research facets), types of STCE activities (e.g., guidelines for development of test code, quality assessment and quality improvement), and types/scales of software systems used for testing and evaluations in the primary studies.

The main contributions of this article are three-fold:

- A systematic map (Section 5) developed for the area of STCE which could be used for further secondary studies (e.g., surveys, systematic mappings or systematic literature reviews) in this field in future.
- Systematic mapping of the existing research in this area (Sections 6).
- An online article repository which has been created during this SM [68].

The remainder of study is organized as follows. Section 2 discusses background and related work. Section 3 describes our research method, including the overall SM process, the goal and research questions tackled in this study. Section 4 discusses the article selection process. Section 5 presents the systematic map which has been built through an iterative selection and synthesis process. Section 6 presents the results of the systematic mapping. Section 7 summarizes the implications of the SM results for researchers and practitioners, and discusses the potential threats to validity of our study. Finally, Section 8 concludes this study and states the future work directions. The reference section at the end of the study is divided into two parts: primary studies of the SM are listed first and then the other references used in this study.

2. Background and related work

This work is a secondary study in the context of automated test-code engineering, and relates to the following areas: (1) Software test automation, (2) STCE process, and (3) Secondary studies in software engineering and software testing.

Since understanding the rest of the paper needs familiarity with the above related areas, we provide a brief background on these concepts in the rest of this section. Sections 2.1 and 2.2 aim at providing unfamiliar readers with a basic knowledge of test automation and test code, and also the process of STCE to better understand the concepts used in our mapping study. For example, terms such as test-code quality assessment in the context of test-code is discussed. Then, since our work is a secondary study, a brief introduction to other secondary studies and systematic mapping studies in software engineering and software testing is provided (Sections 2.3 and 2.4). Related works are discussed afterward (Section 2.5).

2.1. Brief overview of automated testing, test frameworks and test code

Automated testing is the use of special software tool or framework (separate from the software being tested) to test the system under test (SUT).

There are different test tools and frameworks which are used to develop test cases, test code (scripts), manage test cases and control the execution of tests. Test code (script) is a piece of code written in regular programming languages (e.g., Java, C++) which is executed on the SUT for the purpose of testing (exercising) the SUT and observing its behavior/output. A test script usually codes a given test case.

Some software testing tasks, such as extensive regression testing, can be laborious and time consuming to do manually. In addition, a manual approach might not always be effective in finding certain classes of defects. Test automation offers a possibility to perform these types of testing effectively. Once automated tests have been developed, they can be run quickly and repeatedly. Many times, this can be a cost-effective method for regression testing of software products that have a long maintenance life.

By consulting from several books [69–71] on software testing and incorporating different views and classifications, one can divide the testing tasks into five types which we believe is a suitable classification in this context:

1. *Test-case design:* Designing and deriving the list of test cases or test requirements to satisfy coverage criteria (e.g., line coverage), other engineering goals, or based on human expertise (e.g., exploratory testing).
2. *Test scripting:* Documenting test cases in manual test scripts or automated test code.
3. *Test execution:* Running test cases on the software under test and recording the results.
4. *Test evaluation:* Evaluating results of testing (pass or fail), also known as test verdict.
5. *Test-result reporting:* Reporting test verdicts (outcomes) and defects to developers, e.g., via defect (bug) tracking systems.

Each of the above test tasks can be done either manually, fully automated or partially automated. There are many tools and

frameworks for supporting the above test tasks which are either commercial or open-source. Different test tools provide different features for various types of testing (functional versus non-functional testing), and in different test levels (unit, integration or system testing).

The most visible aspect of testing is often the test execution task, in which a test script (code), developed by a human tester or using a tool, is executed on the SUT. A test script usually includes four distinct steps that are executed in sequence:

- *Setup:* The test fixture (the "before" picture) is set up, that is required for the SUT to exhibit the expected behavior as well as anything the tester needs to put in place to be able to observe the actual outcome.
- *Exercise:* The SUT is exercised, i.e., the test code interacts with the SUT.
- *Verify:* Doing whatever is necessary to determine whether the expected outcome has been obtained. This phase determines the outcome of the test case: pass or fail.
- *Teardown:* Tear down the test fixture to put the "world" (SUT and the environment) back into the state in which they were before the test case started execution.

For readers not that familiar with test code, we briefly review next two real examples of automated test code, as shown in Figs. 1 and 2. These test scripts have been developed using two quite-popular test frameworks, JUnit [72] and Selenium [73], respectively. As discussed above, there are many automated test tools and frameworks. Discussing the features of all test tools is outside the scope and need of our study. We are only discussing JUnit and Selenium as two representative examples. JUnit is perhaps the most commonly used unit testing framework [72]. Selenium is a tool [73] for system (functional GUI) testing of web applications.

Fig. 1 is an example test code in unit testing level from the Android software platform (version 2.1) [61,62] which has been written in JUnit. This test method tests whether emergency phone numbers are set properly in the phone under test. The four steps of the test case in Fig. 1 (setup, exercise, verify, and teardown) have been explicitly commented in the code. This unit test code from the Android mobile platform is verifying whether emergency numbers have been properly set on the phone. There are three assertions in the test code of Fig. 1. The first setup and exercise start with populating the mInfo object with the phone number 911,

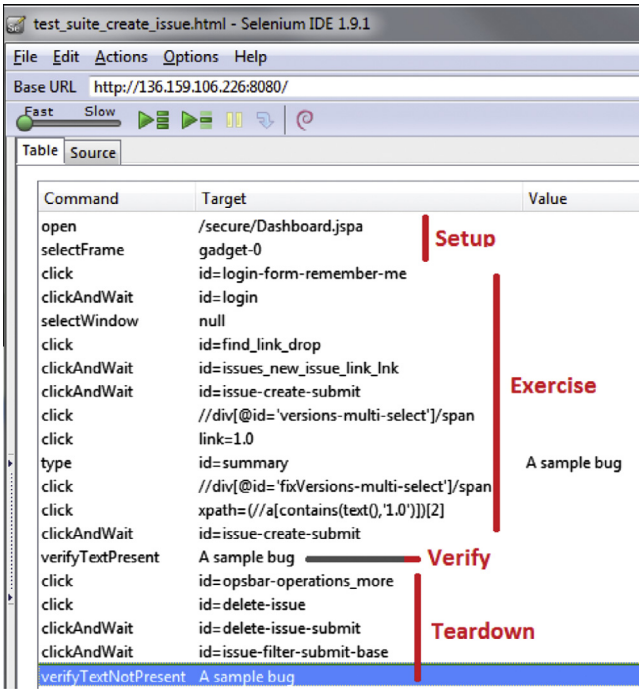


Fig. 2. An example automated system-level test-case code (in tabular form) developed using the selenium tool [73] testing a bug tracking web application.

which is an emergency number in the North America. The first assertion then verifies if the phone number recorded in the mInfo object is an emergency number. The second setup and exercise populate the uniform resource identifier (URI) "tel:911" inside the mInfo object. The follow-up assert method in the verify phase verifies that that URI is also a valid emergency number, as it should be. The last test case starts with a setup and exercise phase which populate the phone number "18001234567" inside the *mInfo* object and then verifies that this number is not an emergency number.

Fig. 2 shows an example automated system-level test-case (in tabular form) developed using the Selenium tool [73] for testing a bug tracking web application called JIRA. This test case tests the enter-an-issue use-case of this system and verifies whether an example bug with summary field = "*A sample bug*" has been

```
/**
 * Checks the caller info instance is flagged as an emergency if
 * the number is an emergency one. There is no test for the
 * contact based constructors because emergency number are not in
 * the contact DB.
 */
public void testEmergencyIsProperlySet() throws Exception {
    CallerInfo mInfo=new CallerInfo();
    // setup and exercise
    mInfo = CallerInfo.getCallerInfo(mContext, "911");
    // verify
    assertIsValidEmergencyCallerInfo();
    // setup and exercise
    mInfo = CallerInfo.getCallerInfo(mContext, "tel:911");
    // verify
    assertIsValidEmergencyCallerInfo();
    // setup and exercise
    mInfo = CallerInfo.getCallerInfo(mContext, "18001234567");
    // verify
    assertFalse(mInfo.isEmergencyNumber());
    // no teardown is needed for this test method
}
```

Fig. 1. An example automated unit test-case code from the Android software platform [61,62].

entered to the database properly. After inserting and testing this example bug, the test case executes the tear-down phase by removing the dummy bug from the system. The four steps of the test script in Fig. 2 (setup, exercise, verify, and teardown) have also been explicitly specified.

2.2. Software test-code engineering

An overview of the STCE process is shown in Fig. 3. There are two domains in the process: the one in the top belongs to the software under test (SUT), also known as "production code", and the one in the bottom is for automated test suites (test code). The process shows the phases involved in development, quality assessment, quality improvement and maintenance of software test code in parallel with software production code.

To conduct automated testing, software engineers have the choice of manually coding the test code, or to use automated tools to generate test code (e.g., using the tool we reported in [55]). In traditional software-development processes such as the Waterfall model, the test suite is developed after the production code of a system is developed. In recent software-development processes such as the Test-Driven Development (TDD) or Test-First Development (TFD), test code is developed first and is used to develop the production code. To account for these different types of development process, the arrow reading "impacts" can be traversed in both directions, meaning that development of production code can impact development of test code, and vice versa.

After the first versions of both (production and test) code bases are ready, both the production code and automated test suites code need to go through quality assessment (verification and validation) activities. Both types of code need to be maintained and evolved together. However, maintenance changes almost always start with (are triggered by) production code. To see an example of how changes in a SUT should be propagated to its test code, let us refer to the example test code in Fig. 2. Let us consider the case when the URL of the web application under test is changed. Note that the test code in Fig. 2 was recorded (encoded) to have /secure/Dashboard.jspa as the starting URL of the web page under test. If the starting URL is changed, the test code will simply not run and has to be revised (i.e., co-maintained) accordingly.

Kaner [74] points out that "*testers invest in regression automation with the expectation that the investment will pay off in due time. Sadly, they often find that the tests stop working sooner than expected. The tests are out of sync with the product. They demand repair. They're no longer helping finding bugs. Testers reasonably respond by updating the tests. But this can be harder than expected. Many automators have found themselves spending too much time diagnosing test failures and repairing decayed tests.*"

In large-scale projects, all the phases of STCE are usually effort intensive. Example case studies on co-maintenance of the source code-bases and their efforts magnitude can be found in [75–78]. Note that some phases of the STCE process shown in Fig. 3 might be bypassed by a given team, e.g., a development team may decide to only develop and co-maintain the test code as the SUT changes, without improving the test-code's quality (e.g., refactoring to keep a high-quality design). However, we should note that the quality of test code is as important as the quality of the application code under test, if not higher.

As discussed in Section 1, after the initial development, like any software engineering artifact, test code requires quality assessment and maintenance. Thus, test scripts have a lifecycle involving different development, quality assessment (verification and validation), and maintenance activities, and must co-evolve with the application code under test. Fig. 4 shows, as a UML state-chart diagram, the life-cycle of a test-code artifact, and intends to complement the process shown in Fig. 3.

The life of a test-code artifact (e.g., a test method in JUnit [72]) starts when it is developed. There are two fundamental reasons for a test-code artifact to be changed (maintained): (1) usually, when the SUT changes, it is needed to update the test code (this is also

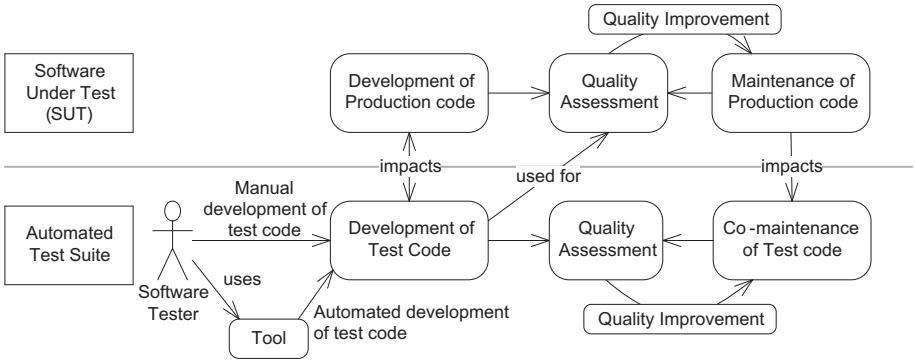


Fig. 3. An overview of Software Test-Code Engineering (STCE) process.

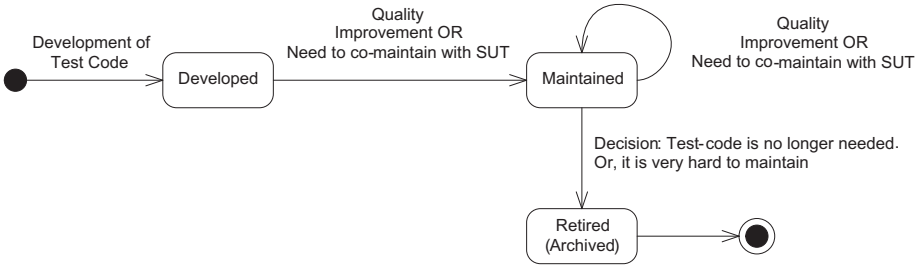


Fig. 4. State-chart diagram of the life-cycle of a test-code artifact.

called "repairing test code" in the literature, e.g., [6,8,30]) and (2) test-code is changed to improve its internal quality (e.g., refactoring), e.g., [3,32,39]. Test-code maintenance could continue and go on for the entire duration of the project and the life-time of the software system. If testing is short phased and if the team decides to no longer use the automated test suites, the test code will be retired/archived. Also, retiring of test code could happen due to unfortunate situation of not being able to co-maintain the test code in a cost effective manner. This unfortunately happens quite often in industrial projects and there are lots of experience reports on failed test automation projects, e.g., [79–81]. Failure of test automation is especially possible when record and playback tools are used for Graphical User Interface (GUI) testing.

In a 2008 book titled "How We Test Software at Microsoft" [37], written by three test managers at Microsoft, the authors discussed and raised the need for tools and infrastructure for test-case and test-code management. Here is a quote from that book: "*When a test team begins writing software to test software, a funny question comes up: "Who tests the tests?" The question is easy to ignore, but it pays to answer it. A tremendous amount of coding effort goes into writing test code, and that code is susceptible to the same sorts of mistakes that occur in product code. In a number of ways, running the tests and examining failures are a test for the tests, but that approach can still miss many bugs in test code.*"

To further highlight the importance of STCE, the book by Microsoft [37] has two specific sub-sections on this topic: "*Section 12.4.4-Test Code Analysis*" and "*Section 12.4.5-Test Code Is Product Code*". In the same book, the authors point out the scale of test code developed by the engineers in Microsoft: "*Systems with a million or more test points (test cases * the number of configurations they run on) are common for Microsoft products*". They also add: "*Product code and test code alike must be highly maintainable*".

The subject of test-script engineering has received the attention of many researchers and practitioners. We are recently seeing focused venues in the area, e.g., the first workshop International Workshop on End-to-End Test Script Engineering (ETSE) [82] which was held in July 2011 and was co-located with the International Symposium on Software Testing and Analysis (ISSTA).

To study test-code maintenance precisely, let us recall from the software maintenance literature that there are four types of software maintenance activities:

- *Perfective:* Improving the quality of software artifacts (e.g., source code) without changing its behavior, e.g., refactoring.
- *Adaptive:* Dealing with changes and adapting in software requirements, or environment.
- *Preventive:* Activities aiming on increasing software maintainability to prevent problems in the future. The decayed parts of the application code are often referred to as *code smells* which are often is symptoms of a problem (e.g., low maintainability). Similarly, a test smell is symptom of a problem in test code.
- *Corrective:* Dealing with errors found and fixing it.

Similar to the maintenance of product code, maintenance of test code (by itself) and also co-maintenance of test code together as the product code is maintained are very important activities [83]. Based on the STCE process depicted in Fig. 3, each of the above types of software maintenance activities, when applied to test code, would be rephrased as follows:

- *Perfective maintenance of test code:* Quality improvement of test code, e.g., refactoring using test-code patterns [27].
- *Adaptive maintenance of test code:* Co-maintenance as the production code is maintained.
- *Preventive maintenance of test code:* Quality Assessment of test code, e.g., detection of test smells in test code, e.g., test smells such as test-code redundancy and eager test [27].
- *Corrective maintenance of test code:* Finding and fixing defects (bugs) in test code.

Through our SM, we will refer to the above different maintenance activities and find studies that have been reported in each of the above categories.

2.3. Secondary studies in software engineering and their usefulness

Research proceeds by learning from and being inspired by existing work. When a research area grows and owns a large number of existing studies, it requires a substantial effort to read all the literature before conducting new research. Summarizing the existing literature and providing an overview for a certain area is helpful for new researchers (e.g., new Master or PhD students), since such summaries identify research trends and shed light on future directions.

Secondary studies are common in software engineering. Secondary study is defined as a study of studies [67], i.e., a review of individual studies (each of which is called a primary study). Example types of secondary studies include: survey studies, systematic mapping (SM) studies, and Systematic Literature Reviews (SLRs). A SM study is a defined method to build a classification scheme and structure a research field of interest. A SLR is a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. SLRs aim to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology. A SLR study often includes a SM study as part of it, and is thus usually more comprehensive than a SM.

Similar to other research fields, software engineering has its methodologies for conducting secondary studies. Petersen et al. [66] presented a guideline study on how to conduct SM studies in software engineering. The guideline study by Petersen et al. [66] provided insights on building classification schemes and structuring a particular sub-domain of interest in software engineering. Kitchenham et al. also presented in [64] detailed guidelines for performing SLR studies in software engineering, most of which could also be used for a SM study. The guidelines described by Petersen et al. [66] and also Kitchenham et al. [64] were followed in our SM study. Justification to follow the guidelines from those two studies is that they are treated as two comprehensive guidelines to conduct SLR/SM in software engineering and have been used by many other researchers conducting and reporting SLRs and SMs in software engineering, e.g., [84–86].

The software engineering community as a whole believes that secondary studies are useful. There are relatively high number of citations to SM and SLR studies, and also there are studies such as [87] which reported the educational value of mapping studies. Last but not least, there has also been a "ternary" study (a study of secondary studies) in software engineering by Kitchenham et al. [65] which systematically reviewed a selected set of SLRs.

2.4. Secondary studies in software testing

We were able to find 25 secondary studies [88–111], [125] reported, as of this writing, in different areas of software testing. We list and categorize these studies in Table 1 along with their study areas. Based on the "year" column, we observe that more and more SMs and SLRs have recently started to appear in the area of software testing. As per our literature search, we were able to find eight SMs and six SLRs in the area, as shown in the table. The remaining 11 studies are "surveys", "taxonomies", "literature

Table 1
The 2012 master thesis reported in [112] systematically mapped 23 testing tools for automated generation of unit-level test code. The major contributions of that research work were: the categorization of the tools and the type of defects that can be caught by those tools. The mapping categorized the tools based on five criteria: (1) language support, (2) availability, (3) domain, (4) testing technique used, and (5) level of effort needed to use the tool. Table 1–25 secondary studies in software testing.

Type of secondary study	Secondary study area	Number of primary studies	Year	Reference
SM	Non-functional search-based testing	35	2008	[88]
	SOA testing	33	2011	[89]
	Testing using requirements specification	35	2011	[90]
	Product lines testing	45	2011	[91]
	Product lines testing	64	2011	[92]
	Product lines testing tools	33	2012	[93]
	Web application testing	79	2013	[110]
	Graphical User Interface (GUI) testing	136	2013	[111]
SLR	Search-based non-functional testing	35	2009	[94]
	Unit testing for Business Process Execution Language (BPEL)	27	2009	[95]
	Formal testing of web services	37	2010	[96]
	Search-based test-case generation	68	2010	[97]
	Regression test selection techniques	27	2010	[98]
	Web application testing	95	2014	[125]
Survey/analysis	Object-oriented testing	140	1996	[99]
	Testing techniques experiments	36	2004	[100]
	Search-based test data generation	73	2004	[101]
	Combinatorial testing	30	2005	[102]
	SOA testing	64	2008	[103]
	Symbolic execution	70	2009	[104]
	Testing web services	86	2010	[105]
	Mutation testing	264	2011	[106]
	Product lines testing	16	2011	[107]
Taxonomy	Model-based GUI testing	33	2010	[108]
Literature review	TDD of user interfaces	6	2010	[109]

reviews", and "analysis and survey", terms used by the authors themselves to describe their secondary studies.

The number of primary studies studied in each study in Table 1 varies from 6 (in [109]) to 264 (in [106]). Our study started with an initial pool of 72 studies and, after applying systematic inclusion/exclusion criteria, includes 60 in the final pool, as described in Section 4.

2.5. Related works (secondary studies in STCE)

Focusing on the area of STCE, we were not able to find any secondary studies closely related to our work. However, we found two studies [112,113] which have systematically mapped [112] and compared [113] unit-level automated test generation tools. In a strict sense, these two studies are secondary studies of test tools and not research studies in the area of testing. Thus, we did not include them in Table 1. However, since they are related to our work to some extent, we discuss them below.

The comparison case-study study reported in [113] compared several unit-level automated test generation tools. The authors created two sets of test suites: one test set contained random test cases and the other contained values to satisfy edge coverage. The results showed that the automatic test-data generation tools generated tests with almost the same mutation scores as the random tests.

3. Research method

In the following, an overview of our research method and then the goal and research questions of our study are presented.

3.1. Overview

As discussed above, this SM is carried out based on the guidelines provided by [64,66]. In designing the methodology for this SM, methods from several other SMs such as [84–86] were also incorporated. The process that lies at the basis of this SM is outlined in Fig. 5, which consists of three phases:

- Article selection (Section 4).
- Development of the systematic map (Section 5).
- Systematic mapping (Section 6).

The SM process starts with article selection from various academic sources. Then, a systematic map is systematically developed. The systematic map is then used to conduct systematic mapping and results are then reported. Details of the above phases are described in Sections 4–6.

3.2. Goal and research questions

There has been very extensive research on theoretical aspects of software testing, but there has been much less work on "test code research", which lies on the practical side of the software testing field. This has been our primary motive and goal of our study. Also, as per our experience, writing test code for testing an application is the most visible aspect of automated testing for many practitioner developers and testers. Similar to the production code (development code), writing a good test code needs a comprehensive engineering process.

The research approach we have used in our study is the Goal, Question, Metric (GQM) methodology [114]. Using the GQM's goal template [114], the goal of this study is to systematically map (classify) the state-of-the-art in the area of STCE, to identify opportunities for future research, and to find out the recent trends and directions in this field from the point of view of researchers and practitioners in this area. We also want to understand how the research space has evolved over time with regards to the above research attributes. Based on the above goal, the following research questions (RQs) are raised.

- *RQ 1 – Mapping of studies by contribution facet:* How many studies present STCE methods, techniques, tools, models, metrics, or processes? Petersen et al. [66] proposed the above types of

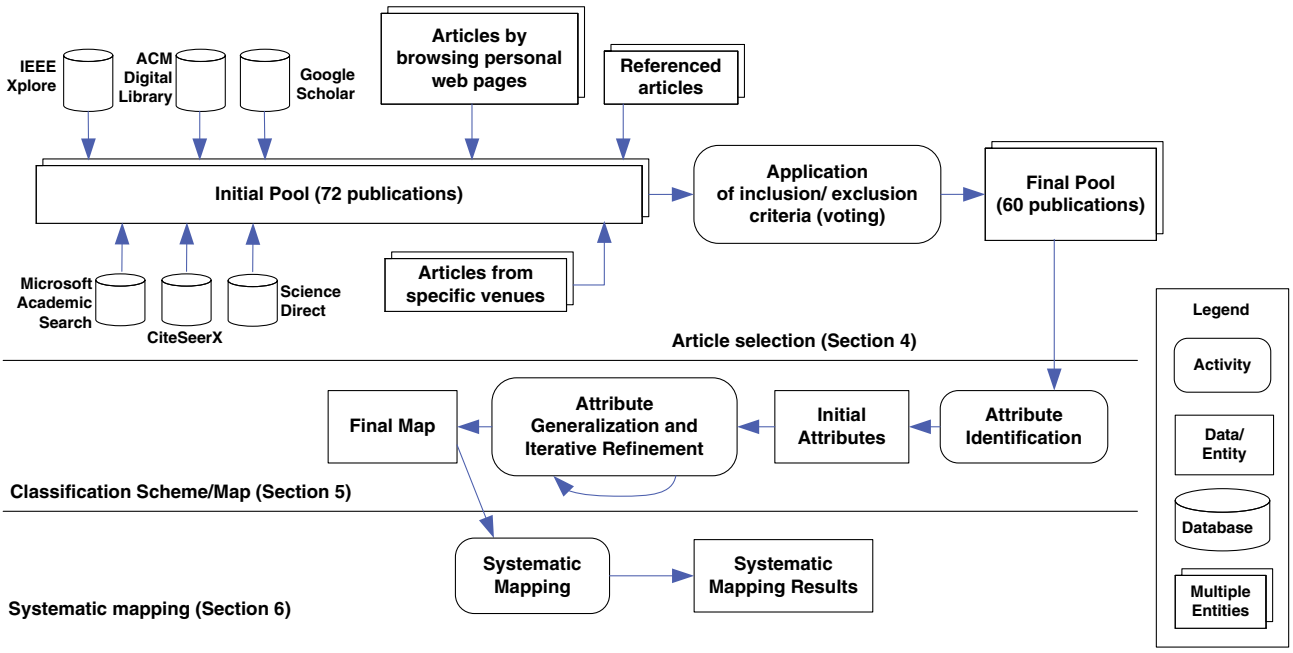


Fig. 5. The research process used to conduct this SM study.

research contributions to enable systematic mapping of studies in software engineering and the above mapping approach has been used in other many other SM studies, e.g., [110,111,115]. Answering this RQ will enable us to assess whether the community as a whole has had more focus on developing new test techniques, or more focus on developing new test tools.

- *RQ 2 – Mapping of studies by research facet:* What type of research methods have been used in the studies in this area? Some studies only propose solutions without extensive validations, while some other studies present in-depth evaluation of their approach (e.g., systematic empirical studies using hypothesis testing). Petersen et al. [66] has also proposed guidelines to classify the research approach of studies, which we will use to answer this RQ. The rationale behind this RQ is that knowing the breakdown of the research area with respect to (w.r.t.) research-facet types will provide us with the maturity of the field in using empirical approaches.
- *RQ 3 – Contribution Facet versus Research Facet:* What ratios of studies fall in each combination of contribution facet and research facets? For example, how many studies have proposed STCE-related metrics with research maturity of systematic empirical studies? Addressing this RQ will help us understand the frequency and ratio of studies for each combination of contribution and research facets.
- *RQ 4 – Types of STCE approach:* What types of STCE activities have been presented in the literature (e.g., development of test code, quality assessment, and quality improvement of test code), and which types are more popular than others? Addressing this RQ will help us gain knowledge about the type of test activities that have been more popular.
- *RQ 5 – Types of STCE activities versus contribution facets:* What ratios of studies fall in each combination of STCE activities and research facets? For example, we were interested in questions such as the following: What ratio of the studies have presented tools for quality assessment of test suites? Addressing this RQ will characterize the research landscape for each combination of STCE activities and contribution facets.
- *RQ 6 – Targets of test-code quality assessment:* What kinds of issues are being addressed in test-code quality assessment? Examples are test smells in test code and test-code maintainability. Addressing this RQ will characterize the research landscape in terms of goals of test-code quality assessment approaches.
- *RQ 7 – Test levels:* Which test levels have received more attention (e.g., unit, integration and system testing)? Addressing this RQ will show the level of focus on each test level and will highlight the open gaps in each test level requiring further work.
- *RQ 8 – Test Frameworks and Tools:* What test frameworks and tools have been used (e.g., JUnit) and what are their ratio? Addressing this RQ will identify the popular test frameworks and tools in the STCE domain.
- *RQ 9 – Software systems under test (SUT):* What are the types, and sizes of the software systems under analysis in the studies? What ratios of studies have used open-source, commercial, or academic experimental systems for evaluation? Addressing this RQ will characterize the scale, size and type of SUTs analyzed in the STCE domain.
- *RQ 10 – Tools presented in studies:* What STCE tools have been proposed in the studies? Are they available for download and/or purchase? Note that this question differs from RQ 8 (test tools "used" in the studies). Addressing this RQ will identify the test tools developed and proposed in the STCE literature.
- *RQ 11 – Academia versus Industry and their Collaborations:* What ratios of the studies are from academia versus industry? What are the most active industrial companies in this area? What are the main focus areas of researchers from academia versus industry? Addressing this RQ will show the scale of academia-industry collaborations in the STCE literature.

4. Article selection

Let us recall from our SM process (Fig. 5) that the first phase of our study is article selection. For this phase, we followed the following steps in order:

Table 2
Search keywords.

Focus	Test code Test script Automated test suite Automated test Automated testing
Type of activity	Development Quality assessment Quality improvement Verification Validation Maintenance Evolution

- Source selection and search keywords (Section 4.1).
- Application of inclusion and exclusion criteria (Section 4.2).
- Finalizing the pool of articles and the online repository (Section 4.3).

4.1. Source selection and search keywords

Based on the SM guidelines [64,66], to find relevant studies, we searched the following six major online search academic article search engines: (1) IEEE Xplore,[1] (2) ACM Digital Library,[2] (3) Google Scholar,[3] (4) Microsoft Academic Search,[4], (5) CiteSeerX,[5] and (6) Science Direct.[6]

In order to ensure that we included as many relevant studies as possible in the pool of selected studies, we identified all potential search keywords regarding the focus of each of our RQs. Using an iterative improvement process, we extracted two lists of search keywords as shown in Table 2: (1) the first group was the focus area of our work, i.e., software test code and (2) second group was about types of activity, e.g., quality assessment and maintenance (of test code). Each item from the first list was combined with each item from the second list. This resulted in $5 \times 7 = 35$ combinations which we searched for. In terms of the search time-window, the searches were conducted in October–November 2012 and thus only studies available in the above search engines by that time were included in our pool.

To decrease the risk of missing relevant studies, similar to previous SM studies and SLRs, we searched the following sources as well manually:

- References found in studies already in the pool.
- *Personal web pages of active researchers in the field of interest:* We extracted the names of active researchers from the initial set of studies found in the above search engines.
- *Specific venues:* They were the venues which ranked the highest (in terms of number of published studies) in the initial set of studies, i.e., we looked at the venues in the initial set of papers and selected the ones which has published most of the studies. The specific venues that we looked into their proceedings were: International Workshop on End-to-End Test Script Engineering (ETSE) which is collocated with International Symposium on Software Testing and Analysis (ISSTA), and International Conference on Software Testing (ICST).

All studies found in the additional venues that were not yet in the pool of selected studies but seemed to be candidates for inclusion were added to the initial pool. With the above search strings and search in specific venues, we found 72 studies which we considered as our initial pool of potentially-relevant studies (also depicted in Fig. 5). At this stage, studies in the initial pool were ready for application of inclusion/exclusion criteria described next.

4.2. Application of inclusion/exclusion criteria

In our study, the following inclusion criteria were considered: (1) relevance of the topic of each study to the STCE concepts, (2) the level of comprehensiveness and evaluation followed in the study, and (3) whether the study was peer reviewed. If multiple studies with the same title by the same author(s) were found, the most recent one was included and the rest were excluded.

Only studies written in English language and only the ones which were electronically available were included. If a conference study had a more recent journal version, only the latter was included. We also included the books which had contents related to STCE, since after reviewing several of them, we found several very good books in this area, e.g., [12,27,37].

The relevance of each candidate study to the STCE was carefully considered. As exclusion criteria, we excluded studies which had addressed test-suite issues from a conceptual (non-implementation-oriented) viewpoint, e.g., set theory, and not directly on test-code level. An example study in this group is [116] which presented techniques for test-suite minimization, selection and prioritization, but there was no focus on test code, rather the discussions were in a conceptual viewpoint, where each test suite is treated like a set and test-code specific challenges were not addressed. Also, studies using or proposing new ideas in mutation testing and code coverage were excluded since those two techniques are used to enhance test suites by adding test cases (e.g., developing new test code) and not analyzing the quality of "existing" test code. For example, studies such as [117,118] were reviewed and excluded since they approached test coverage issues from a non-implementation-oriented viewpoint, and did not address test-code specific challenges in the context of test coverage. Also, papers in other areas of testing, e.g., GUI testing, which had not taken into account test-code specific challenges were not included. However, a GUI testing paper such as [56] was included since it presented an approach for comparing the quality of automated test scripts (oracles) for GUI-based software. Works reported in [119,120] are two other examples of the studies that were excluded, since [120] had focused on test suites from a conceptual viewpoint and not on test-code, and [119] was only a workshop presentation PDF file and we could not find any formal study corresponding to it. We should also mention that the online repository of the papers in our pool (hosted on Google Docs) [68], refer to the tab "Excluded" in the online spreadsheet, contains detailed explanations on why each paper has been excluded from the primary pool.

To apply the inclusion/exclusion criteria to the initial pool, the authors of this article inspected the studies in the initial pool and assigned a vote on a 9-point scale to each study, with '9' indicating a strong opinion in favor of not excluding a study, and '1' indicating a strong opinion in favor of excluding a study. We decided to use a threshold of 5 marks for the decision on study exclusion, i.e., studies with cumulative votes of less than 5 marks were excluded. To vote on each study, we reviewed its title, abstract and keywords. If not enough information could be found in those sources, a more in-depth evaluation was conducted. Based on the results of the joint voting, the size of the pool of selected studies decreased from 72 to 60.

[1] http://ieeexplore.ieee.org.
[2] http://dl.acm.org.
[3] http://scholar.google.com.
[4] http://academic.research.microsoft.com.
[5] http://citeseerx.ist.psu.edu.
[6] http://www.sciencedirect.com

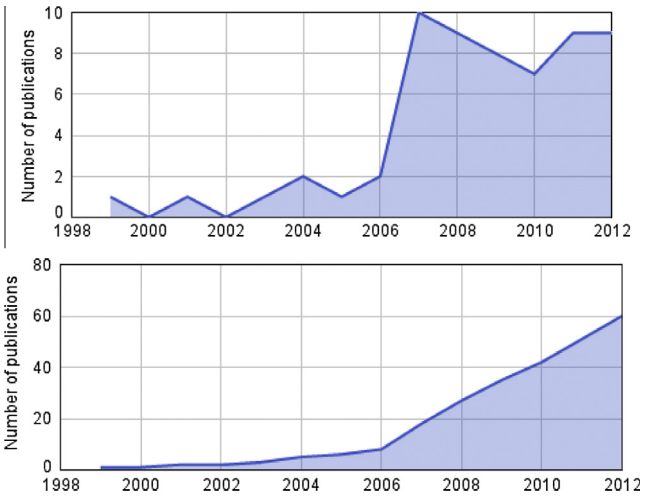


Fig. 6. Annual trend of studies included in our pool. Top: numbers per year. Bottom: cumulative values.

4.3. Final pool of articles and the online repository

After the initial search and the follow-up analysis for exclusion of unrelated and inclusion of additional studies, the pool of selected studies was finalized with 60 studies. The reader can refer to section 'Primary studies' for the full reference list of 60 primary studies. The final pool of selected studies has also been published in an online repository using the Google Docs system, and is publically accessible online at [68]. The classifications of each selected publication according to the classification scheme described in Section 5 are also available in the online repository. We present briefly next the annual trend of studies and the breakdown of studies by venue types.

4.3.1. Annual trend of publications

Fig. 6 shows the annual trend of studies included in our pool. The top chart shows the number per year, and the bottom chart shows the cumulative values. We can observe that the earliest STCE publication appeared in 1999 and is actually a book entitled: *Software Test Automation: Effective Use of Test Execution Tools* [12]. Chapter 7 of this book is related to STCE which is entitled: *Building maintainable tests*.

We can observe from Fig. 6 that the number of studies before 2006 is quite low (less than two per year). But interestingly and surprisingly, from year 2007, there is a sudden increase in the number of studies in the area. This trend denotes the increasing importance and significance of STCE in the software engineering community in recent years.

4.3.2. Publication venues

Fig. 7 shows the number of studies based on their venue types. 34 Studies in the final pool were conference studies, followed by 12 workshop and 7 journal studies. There were 4 books and 3 symposium studies.

The list of venues (e.g., conferences and journals) from which at least two studies were included in our pool is shown in Fig. 8. Full venue names are also shown under the figure. The International Workshop on End-to-End Test Script Engineering (ETSE) seems to be the flag-ship event on STCE, which has started to be held annually since 2011, and co-located with the International Symposium on Software Testing and Analysis (ISSTA). The full list of venues can be found in our online repository [68]. To see all the venue data in detail, the reader is referred to the online repository on the Google Docs [68].

5. Development of the systematic map (classification scheme)

Iterative development of our systematic map is discussed in Section 5.1. Section 5.2 presents the final systematic map. Section 5.3 discusses our data-extraction approach.

5.1. Iterative development of the map

To develop our systematic map, as it is shown in Fig. 5, we analyzed the studies in the pool and identified the initial list of attributes. We then used attribute generalization and iterative refinement to derive the final map.

As studies were identified as relevant to our research project, we recorded them in a shared spreadsheet (hosted in the online Google Docs spreadsheet [68]) to facilitate further analysis. The following information was recorded for each study: (1) study title, (2) publication venue, (3) year of publication, and (4) type of the institution that each author is affiliated with (academic or industry).

With the relevant studies identified and recorded, our next goal was to categorize the studies in order to begin building a complete picture of the research area. Though we did not a priori develop a categorization scheme for this project, we were broadly interested in: (1) types of STCE approaches and (2) types of systems under test which were used in studies.

We refined these broad interests into a systematic map using an iterative approach. The authors of the study consulted with their colleagues and peers to ensure the high quality of the systematic map. The authors conducted an initial pass over the data, and based on (at least) the title, abstract and introduction of the studies, created a set of initial categories and assigned studies to those categories. When the assignment of studies to categories could not be clearly determined just based on the title, abstract and introduction, more of the study was considered. In this process, both the categories and the assignment of studies to categories were further refined.

5.2. Final systematic map

Table 3 shows the final classification scheme that we developed after applying the process described above. In the table, column 1

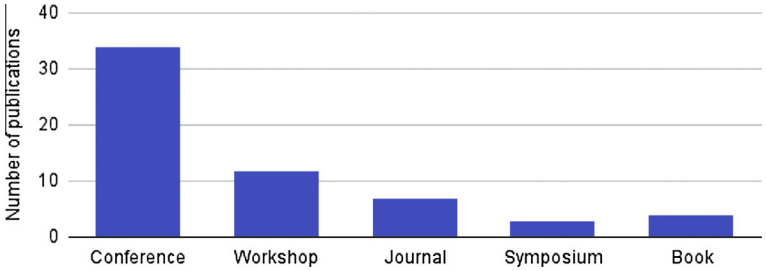
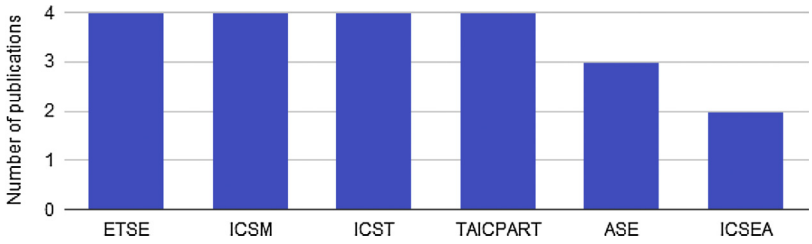


Fig. 7. Number of studies based on their venue types.



Acronym	Full venue name
ETSE	International Workshop on End-to-End Test Script Engineering
ICSM	International Conference on Software Maintenance
TAICPART	Testing: Academic and Industrial Conference
ASE	International Conference on Automated Software Engineering
ICST	International Conference on Software Testing
ICSEA	International Conference on Software Engineering Advances

Fig. 8. Venues with at least two studies included in our pool.

is the list of RQs, column 2 is the corresponding attribute/aspect. Column 3 is the set of all possible values for the attribute. Finally, column 4 indicates for an attribute whether multiple selections can be applied. For example, in RQ 11 (author affiliations), the corresponding value in the last column is 'S' (Single). It indicates that one study can be classified under only one affiliation (e.g., Academia, Industry, Joint work. In contrast, for RQ 1 (Contribution type), the corresponding value in the last column is 'M' (Multiple). It indicates that one study can contribute more than one type of options (e.g., method, tool, etc.).

We utilized the following techniques to derive the list of categories for each attribute: attribute generalization, clustering and aggregation. If there were choices under "Other" which were more than five instances, we grouped them to create new categories. We believe most of the categories in Table 3 are self-explanatory, except for those for contribution and research types (RQ 1 and RQ 2) which are explained in the next two sub-sections.

5.2.1. Type of study: Contribution facet

The first set of categories in our scheme is related to the contribution facet of the study. Petersen et al. [66] proposed the classification of contributions into: method/technique, tool, model, metric and process. These types were adapted in our context. We also added another type: survey or empirical results, since we found that many studies contribute such results. If a study could not be categorized into any above-mentioned types, it would be placed under "Other".

Table 3
Systematic map developed and used in our study.

RQ	Attribute/aspect	Categories	(M)ultiple/(S)ingle
1, 3, 5	Contribution type	{Method (technique), Tool, Metric, Model, Process, Empirical (Case) study, Other}	M
2, 3	Research type	{Solution Proposal, Validation Research, Evaluation Research, Experience Studies, Philosophical Studies, Opinion Studies, Other}	S
4, 5	Type of STCE approach	{Development of Test Code, Quality Assessment, Quality Improvement, Co-maintenance with production code, Repair (a sub-type of Co-maintenance), Other}	M
6	Targets of quality assessment	{Looking for test smells, Quality/adequateness of test oracle (assertion), Other type of QA}	M
7	Test level	{Unit, Integration, GUI/System, Other}	M
8	Test framework/tool used	{JUnit, Other xUnit, TTCN, Other}	M
9	Attributes of the software systems under test (SUT)	Number of software systems: integer SUT names: array of strings Type of system(s) {Academic experimental or simple code examples, Real open-source, Commercial} LOC of system(s): integer LOC of test suites: integer Number of test cases: integer Other size metrics (if provided)	M
10	Attributes of the tool(s) presented in the study	Name: string Available for download (as reported in the study): Boolean URL to download: string	M
11	Author affiliations and active industrial groups	Author Affiliations ∈ {Academia, Industry, Combination (joint work)} Company name: string	S

5.2.2. Type of study: Research facet

The second set of categories in our scheme deals with the nature of the research reported in each study. These categories were influenced by categories described by Petersen et al. [66] and our aim is to provide insights into the empirical foundation being developed by the body of research. The "research type" categories include:

- *Solution proposal:* A study in this category proposes a solution to a problem. The solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution are shown only by a small example or a good line of argumentation.
- *Validation research:* A study in this category provides preliminary empirical evidence for the proposed techniques or tools. These studies either proposed a novel technique/approach and its limited application in a certain context to demonstrate its effectiveness, or conducted a survey or interview among a certain number of participants to answer a particular research question. More formal experimental methods (e.g., hypothesis testing, control experiment) or results are further needed to build relevant theories.
- *Evaluation research:* These studies go further than studies of type "Validation research" by using strict and formal experimental methods (e.g., hypothesis testing, control experiment) in evaluating novel techniques or tools in practice. Hence, these studies provide more convincing empirical evidence and are helpful to build theories.
- *Experience studies:* Experience studies explain how something has been done in practice, based on the personal experience of the author(s).
- *Philosophical studies:* These studies sketch a new way of looking at existing things by structuring the area in form of a taxonomy or conceptual framework.
- *Opinion studies:* These studies express the personal opinion of the author(s) around whether a certain technique is good or bad, or how things should been done. They do not significantly rely on related work or research methodologies.
- *Other:* A catchall category in the event that the work reported in a study does not fit into any of the above research types.

5.3. Data extraction

To extract data, the studies in our pool were reviewed with the focus of each RQ and the required information was extracted. We also ensured to incorporate as much explicit "traceability" links between our mapping and the primary studies as possible. We have put traceability "comments" inside the cell of the online repository [68] where needed to justify why a given mapping was done. For example, the study [29] was categorized for the types development, co-maintenance and repair under STCPE type due to the following quote from this study: "*In this study, we propose an approach for automatically repairing and generating test cases during software evolution*." In those case, the exact quotes have been placed traceability "comments" inside the cell of the online repository [68].

6. Results of systematic mapping

Results of the systematic mapping are presented in this section from Sections 6.1–6.9.

6.1. Mapping of studies by contribution facet (RQ 1)

Fig. 9 shows the annual cumulative breakdown and the total number of primary studies by contribution facet types. Exact study references have also been provided under the figure. The top three contribution facets are: tool, method (also referred to as technique or approach), and metric, which have appeared in 41 studies (68%), 39 studies (65%), and 12 studies (20%), respectively. Note that most of the time-trend charts in this study such as the one shown in Fig. 9 are cumulative stack charts, showing the cumulative number of studies in each year, as accumulated from previous years.

Further note that since many studies were classified under more than one contribution facet, the stack chart of Fig. 9 cannot be used as the annual trend of number of studies (that trend was presented in Section 4.3.1). We can see that across different years, different contribution facets have been studied, and no clear trend change is observable (see Fig. 10).

Based on their contributions, some studies were classified under more than one facet. For example, the work by Van Rompaey and Demeyer [42] makes two contributions: (1) a test tool called *TestView* and (2) a set of visualization models to support test code analysis and quality assessment. Two studies [7,11] have made "other" types of contributions. Ref. [7] is a "meta-analysis", as referred to by its authors, which explores the trade-offs between, and the contrasting characteristics of, the two TTCN-3 sets of refactoring rules. Ref. [11] is a "guideline" for maintainability of TTCN-3 test suites.

Notable findings: We can see that a *healthy* mix of all contribution facets exist in the domain of STCE. It is good news to see a large ratio of test tools among the studies.

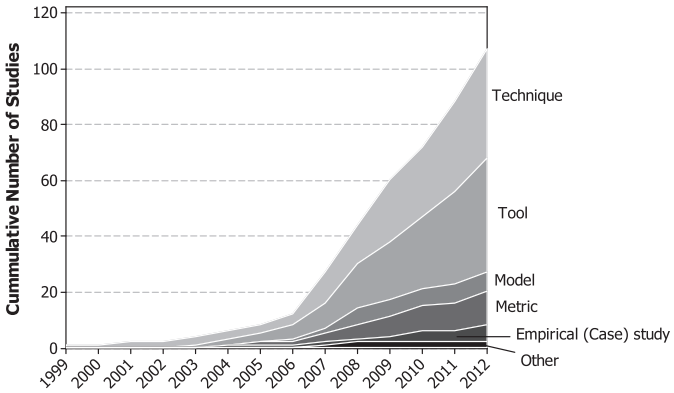
As a point of reference and to put the ratios of number of studies by contribution facet in context, we show the mapping of the web application testing (WAT) studies by contribution facet which has been taken from our other recent SM study in [110]. By comparing the ratios of contribution facet breakdown between the pools of studies in the two domains (WAT and STCE), we can see that there are some similarities and differences. One difference is that there are more tool related papers in the STCE domain compared to WAT. This could be perhaps rooted in the more practical nature of the STCE domain compared to the WAT domain.

6.2. Mapping of studies by research facet (RQ 2)

Based on the scheme described in Section 5, we classified the studies into five research-facet categories. Fig. 11 shows the classification of the all primary studies according to the type of research method they have followed and reported. Recall that, for the research facet type, each study could be categorized under one or more categories.

The ranking of research facets are: validation research (27 studies), evaluation research (16 studies), solution proposal (16 studies), opinion studies (2 studies), and experience reports (2 studies). There was no philosophical study in the pool. Since 27% of the studies have used (systematic) evaluation research approaches, it seems that the STCE field as a whole is leaning more toward to more rigorous validation approaches, which is a good sign.

We also thought it would be useful to compare the research facet breakdown of this SM to three other representative SM studies in software engineering [110,111,115]. We randomly selected three SM studies in software engineering: web application testing [110], GUI testing [111], and Dynamic Software Product Lines (DSPL) [115]. The comparison is visualized in Fig. 12. It is quite interesting that the percentage values for each research facet in the four SM studies are quite closely comparable, with most of



Contribution facet types	Number of studies (%)	References
Method / technique	39 (65%)	[2, 3, 5, 7-10, 12, 15-20, 22, 24-30, 32, 34-39, 43-45, 49, 50, 53, 56, 57, 59, 60]
Tool	41 (68%)	[1-3, 5, 6, 8, 9, 11, 15-21, 25, 26, 28-34, 37-39, 42, 44, 45, 48, 50-55, 57-60]
Model	7 (12%)	[11, 18, 33, 40-42, 57]
Metric	12 (20%)	[2, 13, 20, 22, 23, 31, 33, 34, 40, 43, 45, 52]
Process	2 (3%)	[36, 37]
Empirical (case) study	6 (10%)	[2, 4, 13, 14, 46, 47]
Other	2 (3%)	[7, 11]

Fig. 9. Mapping of studies by contribution facet. (See above-mentioned references for further information.)

the primary studies being mapped as using solution proposals, validation research and then evaluation research.

6.3. Contribution facet versus research facet (RQ 3)

Similar to other SM studies (e.g., [66,115]), we reviewed the number of studies by research facet versus contribution facet. As a bubble chart, Fig. 13 shows that information. As we can observe, a large ratio of the primary studies are positioned in the bottom-left corner. This means that many studies have proposed techniques and tools and have evaluated those in all three levels of research maturity: solution proposal, validation research, and evaluation research. Many intersections have zero values, e.g., there are no studies proposing metrics with the "experience report" research facet, and thus there is a need for further studies in those areas with low attention For example, further work is needed on metric and models in the context of STCE with various research facets.

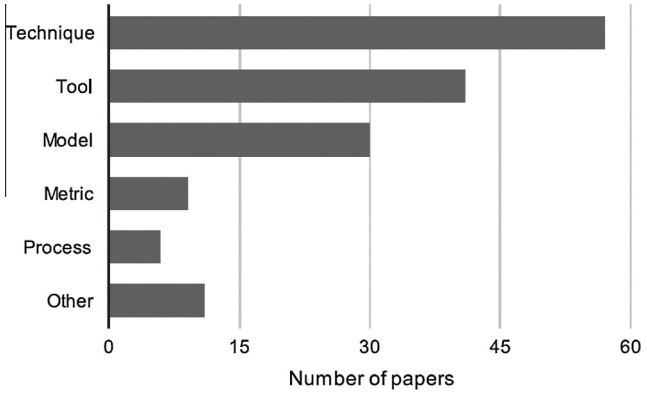


Fig. 10. Mapping of the web application testing (WAT) studies by contribution facet (taken from the SM study in [110]).

6.4. STCE activities

We present next the results of the SM for RQs 4–6.

6.4.1. Types of STCE activities (RQ 4)

Fig. 14 shows the mapping of the studies in terms of the STCE activities they have proposed. Both cumulative and also yearly numbers are shown. As we can see, the top three STCE activities in order are: (1) quality assessment, (2) co-maintenance, and (3) quality improvement and test-code development. Studies working on co-maintenance of test code have had the highest growth rate in recent years, denoting the importance and also the challenges involved with this specific STCE activity. Again note that since a given study could be classified under more than one STCE activity, the cumulative number at the end of the time period (i.e., the value of 82) surpasses the number of studies in the pool (i.e., 60).

Five works [3,16,19,21,54] have reported "Other" types of STCE activities. Coincidentally four of them have proposed techniques for test-code comprehension (understanding).

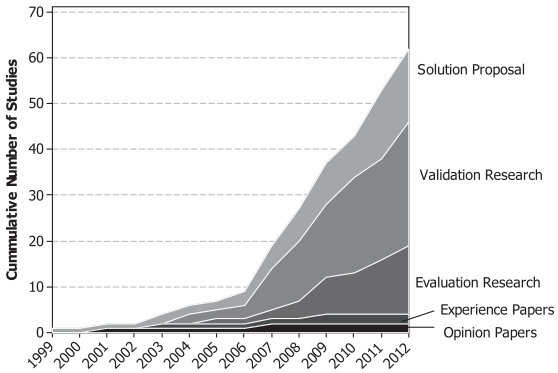
Since a single study could present more than one type of STCE activity, we counted the number of STCE activities presented in one single publication and the resulting histogram is shown in Fig. 15. We can see that most of the studies (34 of them) presented one type of STCE activity, while 21 and 4 studies presented two and three types of STCE activities, respectively.

Notable findings: Based on the annual trend of STCE activities (in Fig. 14), we can see that, starting from year 2007, there is a sharp increase in the number of studies especially in the area of test quality assessment.

6.4.2. Types of STCE activities versus contribution facet (RQ 5)

Similar to several other SM studies (e.g., [115]), we also counted the number of studies for each pair of contribution facets types and STCE activity types. We were interested in questions such as the following: What ratio of the studies have presented tools for quality assessment of test suites?

As a bubble chart, Fig. 16 shows that information. As we can observe, a large ratio of the primary studies are positioned in the bottom-left corner. Many studies have proposed techniques and tools for quality assessment and co-maintenance of test suites, compared to other types of combinations. This seems to denote the importance and also the need for tool-support in those activities. Also, Fig. 16 denotes that there is a need for further work in the working areas:



Research facet types	Number of studies (%)	References
Solution proposal	16 (27%)	[1, 5, 11, 12, 17, 18, 25-28, 35, 36, 44, 45, 53, 54]
Validation research:	27 (45%)	[3, 4, 6, 9, 20-24, 29, 30, 32, 33, 37, 39, 40, 42, 46-48, 50-52, 55, 57-59]
Evaluation research	16 (27%)	[2, 8, 13-16, 19, 31, 34, 38, 41, 43, 45, 49, 56, 60]
Experience studies	2 (3%)	[37, 44]
Opinion studies	2 (3%)	[7, 10]

Fig. 11. Mapping of studies by research facet. (See above-mentioned references for further information.)

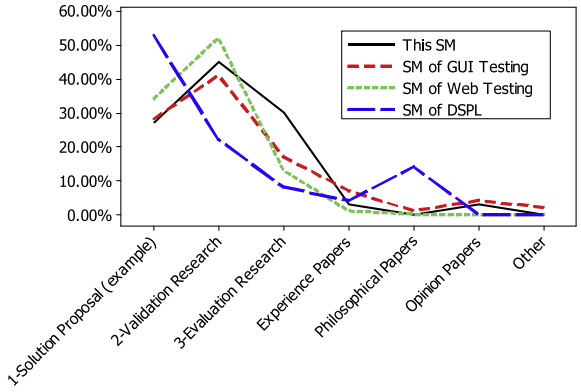


Fig. 12. Comparing the research facet breakdown of this SM to three other representative SM studies [110,111,115].

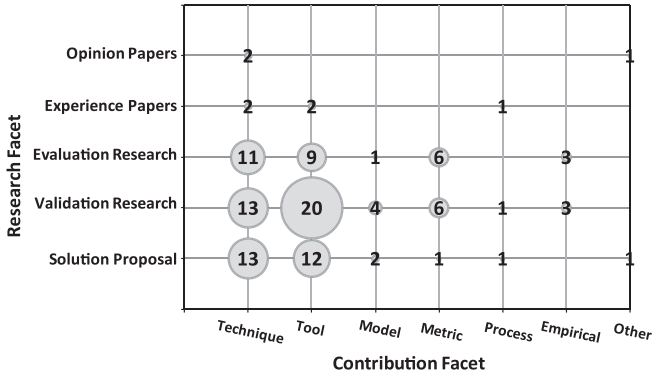


Fig. 13. Number of studies by research facet versus contribution facet (as a bubble chart).

- Model-based approaches for test code repair, co-maintenance and also other STCE activities.
- Further metrics for test code repair, and co-maintenance.
- Processes for test code repair, and co-maintenance.

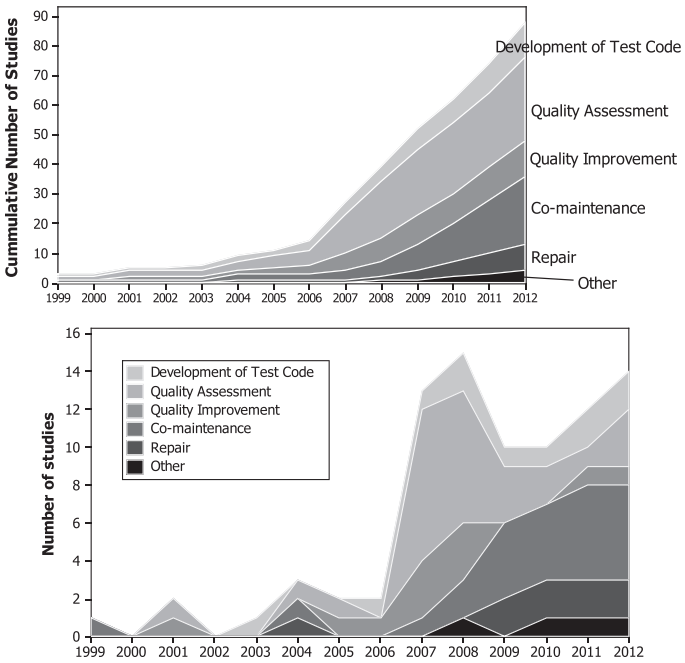
6.4.3. Targets of test-code quality assessment (RQ 6)

As discussed in Section 6.4.1, nearly half of the studies (28 studies, or 47%) have focused on quality assessment. We wanted to determine the goals (targets) of quality-assessment in those studies. As discussed in the final systematic map, the possible categories were: looking for test smells, quality/adequateness of test oracle (assertion), and other types of QA.

17 Of those 28 studies investigated how to detect test smells. For example, Ref. [33] presented an instantiation of the software quality model proposed by the ISO/IEC standard 9126 for TTCN-3 test scripts as well as an approach to assess and improve the quality of TTCN-3 test specifications. The assessment is based on metrics and the identification of test code smells. Ref. [2] presented an empirical analysis of the distribution of unit test smells (as threats to test code quality) and their impact on software maintenance. The work reported in Ref. [5] identified the bad test smells and proposed guidelines for test-code refactoring aiming at removing test smells in the context of GUI test scripts. We observe that the detection of test-code smells is an active area of research.

Six studies [12,23,45,48,52,56] assessed the quality (i.e., adequateness) of test oracle (assertions) in test code. For example, [12] mentioned guidelines on when and how to compare outcome of a test case. Refs. [23,52] focused on the issue of state coverage in test code which measures the adequacy of checks of program behavior by considering the ratio of state updates that are used by oracle assertions to the total number of available state updates. Ref. [45] introduced check coverage, as an indicator of oracle quality which measures oracle adequateness by ratio of statements that contribute to computation of values which get asserted by test cases. Ref. [48] developed an Eclipse plugin for JUnit test cases which uses static analysis of the class under test and already written unit test cases to recommend test-code pieces for developers to put in test oracles or test inputs. Finally, Ref. [56] presented an approach for comparing the quality of automated test oracles for GUI-based software applications.

Also, six studies [11,12,26,31,34,42] conducted other types of quality-assessment on test code which are discussed next. The studies in the "Other" category were as follows: Ref. [11] investigated the understandability, learnability, analyzability, changeability of test code. Ref. [12] discusses efficiency, reliability, flexibility, usability, and portability in addition to its main topic



	Number of studies (%)	References
Development of test code	12 (20%)	[12, 18, 25, 27, 29, 36, 37, 44, 50, 51, 55, 57]
Quality assessment	28 (47%)	[1-3, 5, 10-12, 20-24, 26, 27, 31-34, 37, 39, 40, 42, 45, 48, 52, 53, 56, 58]
Quality improvement	12 (20%)	[1, 5, 7, 10, 17, 27, 31, 33, 34, 36, 37, 49]
Co-maintenance with production code	23 (38%)	[2, 4, 6, 8, 9, 12-15, 28-30, 35, 38, 40-43, 46, 47, 54, 59, 60]
Repair	9 (15%)	[6, 8, 9, 14, 28-30, 47, 59]
Other	5 (8%)	[3, 16, 19, 21, 54]

Fig. 14. Types of STCE activities proposed in the studies. (See above-mentioned references for further information.)

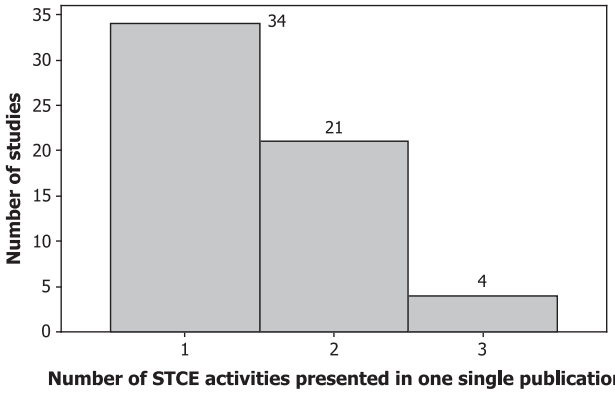


Fig. 15. Histogram of number of STCE activities presented in one single publication.

maintainability of automated test-ware. Ref. [31] assessed quality of test code using metric values compared to similar projects. Ref. [34] also conducted metric-based maintainability analysis. Last but not least, Ref. [26] proposed an approach for ensuring that test code refactoring has been done properly, in the context of test suites written using the TTCN-3 language. Ref. [42] conducted test-code visualization for the purpose of ensuring test-code quality (see Fig. 17).

6.5. Test levels (RQ 7)

We used a multi-choice attribute to classify the types of test levels used in each study: unit testing, integration testing and GUI system testing. Fig. 18 shows the breakdown and the trend over the years.

Notable findings: We can observe that STCE in the context of unit testing is more active compared to integration and system testing. This might be due to existence of more challenges in developing and maintaining unit test code.

6.6. Test frameworks and tools used (RQ 8)

The trend of test frameworks and tools used in the studies are shown in Fig. 19. Note that this question differs from RQ 10 (test tools "presented" in the studies) which will be discussed in Section 6.8.

Being used in 30 studies, JUnit is the clear leader. Other xUnit frameworks (e.g., CPPUnit and NUnit) were used in 15 studies. 8 Studies were using the TTCN notation/framework. 15 Studies used other test frameworks and tools. For example, [46] conducted a study for maintenance of GUI test code written using the IBM Rational Functional Tester tool. Ref. [14] used the HP QuickTest Professional (QTP) tool. Ref. [6] used the Selenium tool.

Notable findings: JUnit has been used in 50% of the studies, showing the popularity and wide usage of this tool in both academia and industry. This might be due to the fact that it has been one of the (or the) earliest unit testing frameworks.

6.7. Software systems under test and test suites (RQ 9)

We discuss next the types, numbers, scale and size of systems under test (SUT) and also test suites used in the studies.

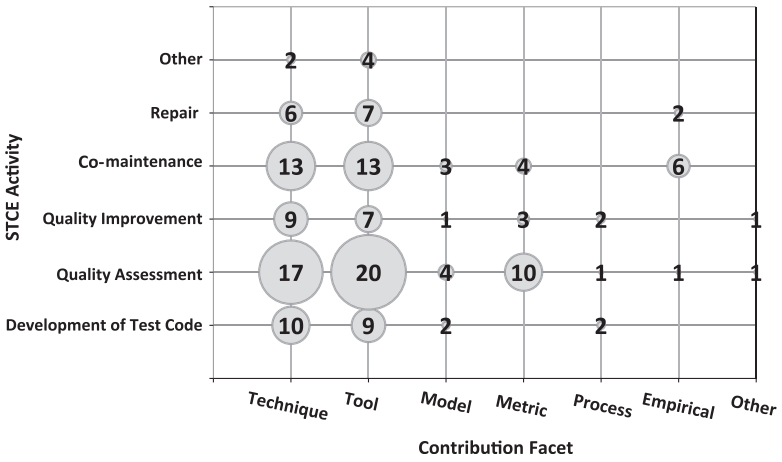
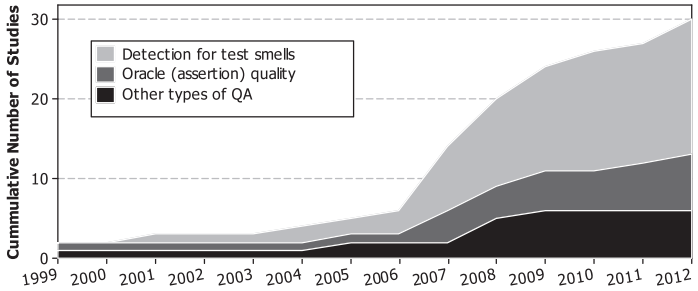
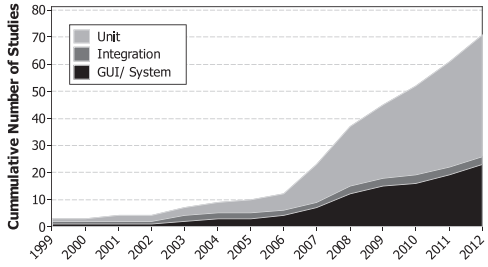


Fig. 16. Number of studies by types of STCE activities versus contribution facet (as a bubble chart).



	Number of studies (%)	References
Detection for test smells	17 (61%)	[1-3, 5, 10, 20-22, 24, 27, 32, 33, 37, 39, 40, 53, 58]
Oracle (assertion) quality/ adequateness	6 (21%)	[12, 23, 45, 48, 52, 56]
Other types of QA	6 (21%)	[11, 12, 26, 31, 34, 42]

Fig. 17. Targets of test-code quality assessment. (See above-mentioned references for further information.)



	Number of studies (%)	References
Unit testing	46 (77%)	[2, 3, 7-13, 16, 17, 19-25, 27-31, 33, 34, 36-45, 47-49, 51, 52, 54, 55, 57-60]
Integration testing	4 (7%)	[12, 37, 44, 60]
System testing	23 (38%)	[1, 4-7, 11, 12, 14-16, 18, 26, 32-35, 37, 44, 46, 47, 50, 53, 56]

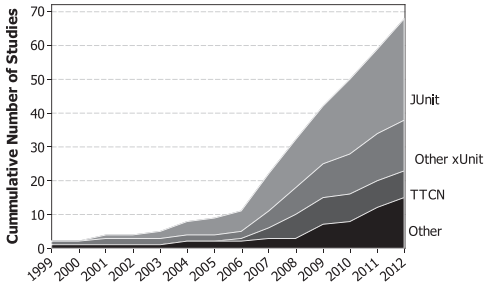
Fig. 18. Types of test levels used in the studies. (See above-mentioned references for further information.)

6.7.1. Software systems under test

We classified the primary studies by the type/scale of the software systems under analysis. The possible categories were: academic experimental systems (or simple code examples), real open-source and commercial systems. Results are shown in Fig. 20. For the purpose of comparison, data from another recent SM study in the area of web application testing [110] for this attribute has also been shown.

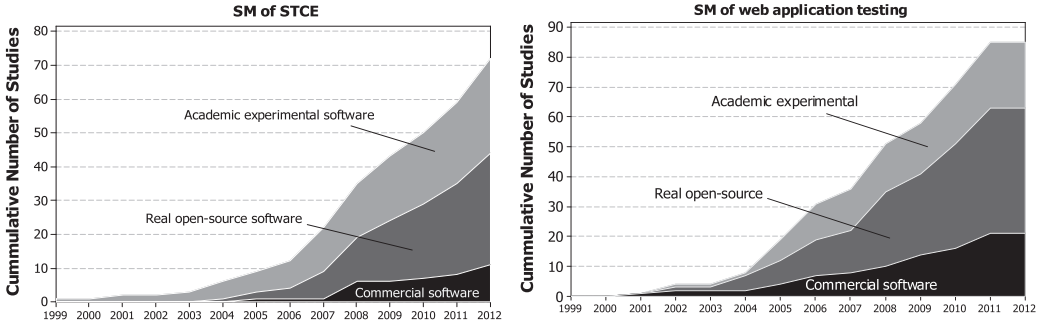
It is clear that real open-source systems are the majority (used in 34 studies or 57%). 27 Studies used experimental systems developed in the academia or simple code examples. 12 Studies used commercial software for evaluation of their methods. It is interesting that the two trends shown in Fig. 20 (the current SM and the web application testing SM [110]) are quite similar.

A number of open-source software systems have been used in the studies for evaluating the STCE approaches, e.g., ArgoUML



	Number of studies (%)	References
JUnit	30 (50%)	[2, 3, 8-10, 13, 16, 17, 20-23, 27-31, 38, 40-45, 47, 48, 57-60]
Other xUnit	15 (25%)	[3, 8, 10, 12, 24, 27, 36, 37, 39, 41, 49, 51, 52, 54, 55]
TTCN	8 (13%)	[1, 7, 11, 26, 32-34, 53]
Other	15 (25%)	[4-6, 12, 14, 15, 18, 19, 25, 35, 37, 46, 47, 50, 56]

Fig. 19. Test frameworks and tools. (See above-mentioned references for further information.)



	Number of studies (%)	References
Academic experimental / simple code examples	27 (45%)	[1, 5, 7, 9, 10, 12, 13, 16, 17, 21, 24-27, 31, 35, 36, 41, 42, 44, 47, 53, 54, 56-59]
Open-source	34 (57%)	[2, 3, 6, 8, 9, 11, 14-16, 19, 20, 22, 23, 28-34, 38-40, 42, 43, 45, 46, 48-50, 52, 57, 58, 60]
Commercial	12 (20%)	[2, 4, 11, 18, 31, 33, 34, 37, 50, 51, 55, 60]
No software under test	0 (0%)	

Fig. 20. Type/scale of the software systems under test. (See above-mentioned references for further information.)

(used in [60]), Session Initiation Protocol (used in [32–34]), and JFreeChart (used in [8,29,38]).

The following commercial software systems or protocols have been used in the studies: a supervisory control and data acquisition (SCADA) system called *Rocket* (used in [55]), standardized test suite of IPv6 (used in [11,32,34]), two unnamed enterprise applications developed in IBM (used in [50]), and Adobe InDesign tool (used in [4]).

Last but not least, the following academic experimental systems have been used in the studies: an experimental ATM machine simulation tool (used in [13]), an experimental Hotel web service (used in [53]), and an experimental car gear-box control software named *EmbeddedGear* (used in [54]).

Notable findings: It is a good sign to observe that many real open-source and also commercial systems have been used as objects of study in many studies in this area.

6.7.2. Test suites used in the studies

Many studies used several SUTs as their object of studies. We extracted the number of SUTs used in each publication. The histogram of those data is shown in Fig. 21. The average number of SUTs in each publication is 3.17. The bar with number of SUTs equal to zero corresponds to the studies which only used simple code examples instead of a software system for evaluations. Most of the studies (16 of them) used only one SUT for evaluation. The publication with the highest number of SUTs was [30] which used 22 SUTs selected from the open-source projects of the Apache Software Foundation.

We also wanted to analyze the code-base sizes of the systems and the test-suite code-bases used. This analysis could show the size scale of the systems used in the studies. We extracted the LOC data from the studies only when they were explicitly reported by the authors, i.e., we did not search the Internet to find the LOC size of a given system if its size was not reported in a given study. Out of the total 60 studies in the pool, only 22 and 16 studies explicitly provided production-code and test-code LOC sizes. For many studies which had used several SUTs, we simply added their LOC sizes together to denote the total LOC assessed in each publication.

In this context, we hypothesized that the LOC of SUTs may have been increasing in newer studies. To visually assess this hypothesis, Fig. 22 shows, as a scatter plot, the LOC sizes versus the years of studies. Each point corresponds to a publication. For better presentation purposes, the y-axis is in logarithmic scale. The correlation value of the production-code and test-code data sets with the year values are only 0.32 and −0.09 respectively, meaning that there is only a weak correlation between the year of publication and production-code SUT size and there is no correlation between year of publication and test-code SUT size. It is still nice to observe that larger and larger SUT sizes appear in more recent studies. Ref.

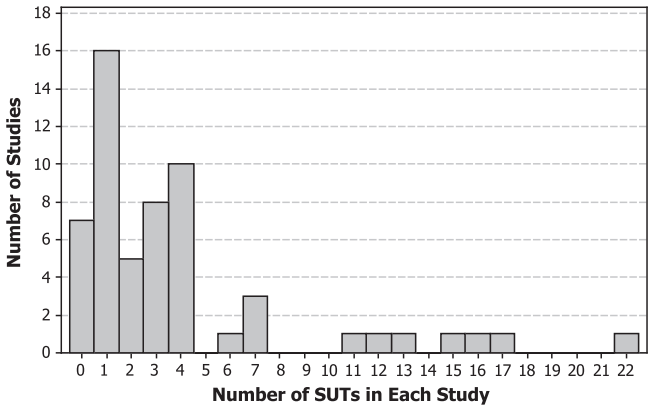


Fig. 21. Histogram of number of SUTs used in each publication.

[29] ranks first with production-code LOC of 1,459,929 which is sum of LOC of 14 SUTs used in that study. In the 2nd rank, study [2] analyzed 18 SUTs with total LOC of 915,000.

It should be noted in this context that most of the studies have not applied their testing technique to the entirety of each SUT that they selected, but only to a few selected sub-systems of each SUT. Thus, the LOC sizes of the SUTs are not entirely precise in terms of the scale of the system and evaluation conducted in each study.

The other size metric that many studies (25 of them) reported was the number of test cases analyzed, used or generated in studies. As a scatter plot, Fig. 23 shows that information. The minimum number of test cases is 23 (reported in [50]) and the maximum is 24,682 (reported in [57,58]).

The last analysis in this sub-section is regarding the following derived metric: (test-code LOC)/(production-code LOC). From size point of view, this metric shows the extent of automated testing for a SUT. In only 11 studies, both the above values were reported, and thus, only for these 11 studies this derived metric could be calculated. As a scatter plot, Fig. 24 shows the ratio of test-suite LOC to SUT size LOC. The maximum and minimum values are 1.61 (reported in [54]) and 0.07 (reported in [2]), respectively.

Notable findings: Different studies have evaluated SUTs with various sizes and scales. It is a positive observation that many large-scale SUTs have been evaluated.

6.8. Test tools (RQ 10)

Forty-one (41) studies presented new tools or extended existing tools (e.g., by developing plug-ins). Note that if a publication had only "used" an existing tool (e.g., IBM Rational Functional Tester), we did not count it in this category. There were cases when a single tool was presented in one study (e.g., TRex in [1]), and it was then extended in several follow-up studies [1,26,32–34]. The list of tools presented (for the first time) or extended in the studies is shown in Table 4. Five publications [2,18,45,50,52] proposed tools with no names.

We thought that an important question to ask in this context is whether the presented tools are available for download, so that other researchers or practitioners could use them as well. We only counted a presented tool available for download if it was explicitly mentioned so in the study. If the authors had not explicitly mentioned that the tool is available for download, we did not conduct

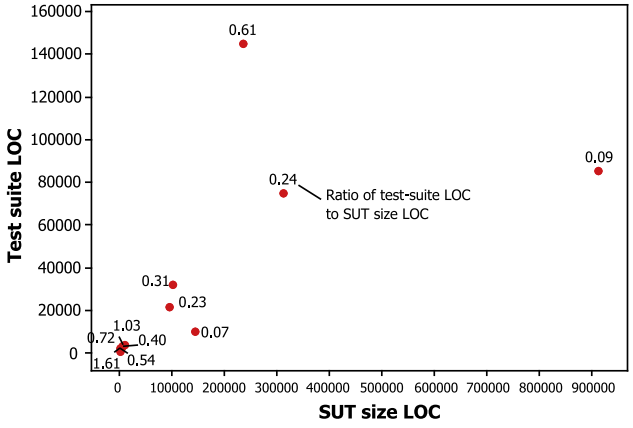


Fig. 24. Individual-value plot of size of systems under test and test suites.

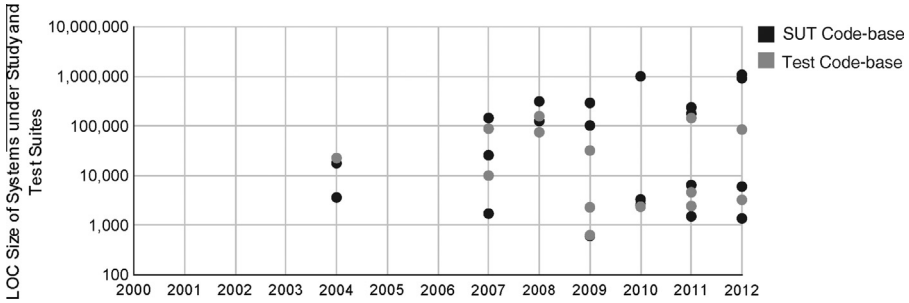


Fig. 22. Size SUT and also test-suite code-bases used in studies.

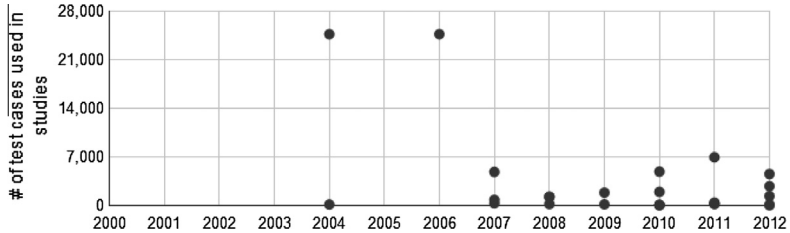


Fig. 23. Number of test cases used in studies.

Table 4
STCE tools presented and extended in the studies (sorted alphabetically).

Tool name	Studies	URL for download
AutoBBUT	[55]	http://www.code.google.com/p/autobbut
AutoETF	[54]	http://www.softqual.ucalgary.ca/tools/AutoETF
GERT	[31]	http://www.gert.sourceforge.net
GTT	[5]	Not provided
KLOVER	[25]	Not provided
Meta-mutation	[19]	Not provided
MODEST	[44]	Not provided
Orstra	[57]	Not provided
Pex	[51]	http://www.research.microsoft.com/en-us/projects/pex
ReAssert	[8,9]	http://www.mir.cs.illinois.edu/reassert
REST	[15]	http://www.cs.uic.edu/~drmark/rest.htm
Rostra	[58]	Not provided
Spectr	[59]	Not provided
TeCReVis	[20,21]	http://www.softqual.ucalgary.ca/tools/TeCRevis
TeMo	[60]	http://www.swerl.tudelft.nl/testhistory
Test Similarity Correlator	[16]	http://www.swerl.tudelft.nl/bin/view/Main/TestSimilarityCorrelator
TestCareAssistant	[28–30]	Not provided
TestEvol	[38]	http://www.cc.gatech.edu/~orso/software/testevol/testevol.html
TestLint	[39]	http://www.typemock.com/test-lint
TestQ	[3]	https://code.google.com/p/tsmells
TestRefactoting	[17]	http://www.sourceforge.net/projects/testrefactoting
TestView	[42]	http://www.lore.ua.ac.be/Research/Artefacts/TestView
TRex	[1,26,32–34]	http://www.trex.informatik.uni-goettingen.de/trac
TTworkbench	[11,53]	http://www.testingtech.com/products/ttworkbench.php
UnitPlus	[48]	Not provided
WATER	[6]	Not provided
Tools with no names	[2,18,45,50,52]	

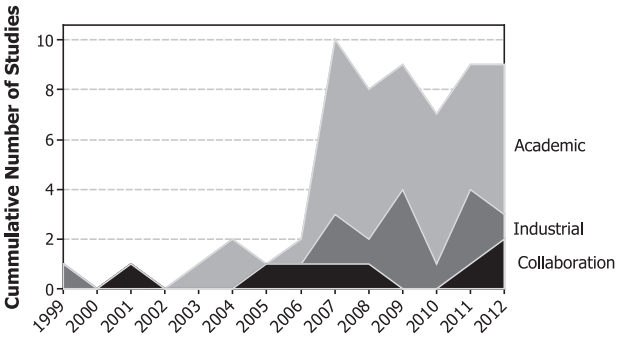


Fig. 25. Author affiliations.

Internet searches for the tool names. Slightly less than half (19 of the 41) tool-presenting studies explicitly mentioned that their tools are available for download.

6.9. Academia versus industry and their collaborations (RQ 11)

For RQ 11, we raised the following three sub-RQs, which are discussed next.

- What ratios of the studies are from academia versus industry?
- What are the most active industrial companies in this area?
- What are the main focus areas of researchers from academia versus industry?

6.9.1. Author affiliations and active industrial groups

As discussed in Section 5.1., we classified the each study in terms of its author affiliation in either of the three following categories: academia, industry, or a joint work (collaboration). Fig. 25 shows the annual trend of that data. In total, there were 39 studies from academia, 13 from the industry, and 8 joint works. As we can observe in Fig. 25, the number of industrial and joint works are slowly starting to rise in recent years which is a positive observation.

6.9.2. Active companies in the industry

To identify the active companies in the area of STCE, we extracted the names of the companies from the pool of industrial and joint works (21 studies in total). Results are shown in Table 5. Microsoft Research with five studies is the most active company in this area. IBM Research, Motorola Labs and Accenture each with two studies stand next.

6.9.3. Focus areas of each group

We clustered the mapping data based on author affiliations versus different type of STCE activities. Results are shown in Fig. 26. We observe that authors from academia and industry have slightly different focuses on different STCE activities.

In terms of priority, academics mostly focus on quality assessment and co-maintenance, while researchers from the industry have focused on development of test-code and then co-maintenance. As for the joint works, they have been mostly on quality assessment and improvement.

7. Discussions

Section 7.1 discusses the summary of findings, summary of trends, and implications of our SM. Section 7.2 discusses the suggested follow-up research directions for the research community. Section 7.3 discussed potential threats to the validity of our study and steps we have taken to minimize or mitigate them.

7.1. Summary of findings, trends, and implications

Based on the data collected for this study, the increase in number of studies (especially from year 2007) indicates the importance of STCE in software research community. This motivated us to perform a systematic mapping for this field. This systematic mapping study, answers 11 research questions (RQ 1–RQ 11) about

Table 5
Companies and industrial research groups active in STCE.

Company name	Number of STCE studies	Studies
Microsoft Research	5	[31,37,49,51,52]
IBM Research	2	[38,50]
Motorola Labs	2	[1,34]
Accenture	2	[14,15]
Agitar	1	[23]
Fujitsu	1	[25]
Grove Consultants	1	[12]
MR Control Systems Inc.	1	[55]
Prodyna GmbH	1	[32]
Sabre Holdings	1	[35]
Software Improvement Group B.V.	1	[10]
Tata Research	1	[18]
Independent consultants	2	[27,36]

different aspects of STCE. Below we summarize the main results of each RQ and discuss the implications of the findings for the research community and also practitioners:

- *RQ 1 (mapping of studies by contribution facet):* There is a mix of papers with respect to different contribution facets in the field of STCE and the top two leading facets are tool (68%) and method (65%). The studies that presented new processes, however, had a low rate (3%), which denotes the need for more process-related studies in this area. By comparing the ratios of contribution facet breakdown between the pools of studies in the two domains of web application testing (WAT) and (STCE), one difference is that there are more tool related papers in the STCE domain compared to WAT. This could be perhaps rooted in the more practical nature of the STCE domain compared to the WAT domain. In general, the reason behind the two leading contribution facets (tool and method) is due to the practical need for tools and methods in this area.
- *RQ 2 (mapping of studies by research facet):* Results of investigation about research facet of studies in RQ 2 and comparing our result to other SM studies shows that similar to other fields in software engineering, STCE is also moving toward more rigorous validation approaches. About 72% of studies belong to the two types of research: validation (45%) and evaluation researches (27%). This result confirms the empirical maturity of STCE field. Philosophical and opinion studies are of less interest which is normal for STCE field; as it should to be more practical and applicable in industrial practices.
- *RQ 3 (contribution facet versus research facet):* Cross analysis of contribution and research facet in RQ 3 helps to reach a better overview about distribution of the STCE studies. The result indicates that not only are tool and method the leading contribution facets, but also they have been validated rigorously. There are still quite a number of solution proposals in tools and methods that can be investigated empirically in future research. In addition, further work is needed on metric and model in the context of STCE with various research facets.
- *RQ 4 (types of STCE approach):* RQ 4 revealed that a good mixture of STCE activities has been presented in the primary studies. Among them, the two leading activities are quality assessment and co-maintenance of test-code with production code. The highest growth rate for co-maintenance activities in recent years shows the importance and challenges involved in this activity.
- *RQ 5 (types of STCE activities versus contribution facets):* The answer to RQ 5 shows that usage rate of tools and techniques among all STCE activities are almost similar. Most of tools and techniques are proposed for quality assessment and co-maintenance. This denotes the importance of tool support in these activities. Models have been used in all activities other than repairing. Also, as depicted in Fig. 16, there is a need for further work in the areas: (1) model-based approaches for test code repair, co-maintenance and also other STCE activities, (2) further metrics for test code repair, and co-maintenance, and (3) processes for test code repair, and co-maintenance. Also, it can be concluded that most empirical studies are dedicated to co-maintenance. Other STCE activities still need more attention regarding empirical studies.
- *RQ 6 (targets of test-code quality assessment):* Answering RQ 6 shows that there are two main categories of quality assessment activity: detection of test smells and oracle assertion adequacy. Test smells can have negative impact on quality of test suites and the majority of studies (61%) in quality assessment belong to identifying test smells. About 21% of studies assess the test suite quality by checking the adequacy of oracle assertions. Other types of quality assurance activities (21%) have focused on assessing different quality characteristics of test codes, such as test-code maintainability, understandability, and efficiency.
- *RQ 7 (test levels):* Analyzing the result of RQ 7 indicates that the most active area with respect to test level is unit testing (77%) followed by system testing (38%). High rate of studies and sharp increase in the number of studies show the importance and challenges in developing and maintaining unit test codes. Integration testing is one of necessary levels of testing; however, there were only few studies (7%) devoted to it in the context of STCE. As per our experience, most of the open-source systems have either unit and/or system test suites. There is a general lack of case studies and availability of their integration test suites. Integration testing is often conducted using unit test framework, e.g., JUnit. There are not many tools for integration testing. Thus, we can say that there is a need for more work focusing on integration testing.
- *RQ 8 (test frameworks and tools):* RQ 8 revealed that JUnit is the leading test framework which has been used in about 50% of the studies. We think that it is because many open-source systems already have existing test suites in JUnit and this makes

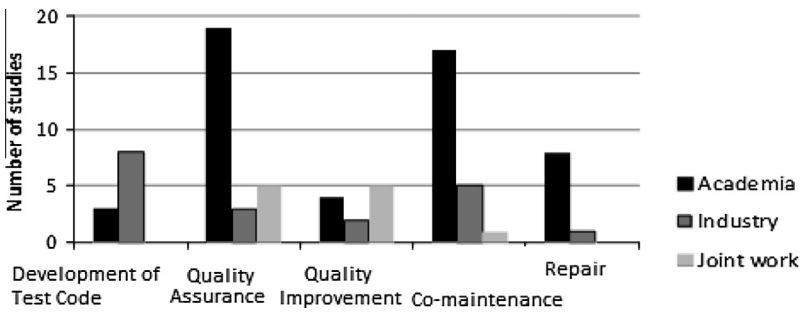


Fig. 26. Focus of different author groups on different STCE activities.

conducting case studies using JUnit-based test suites convenient. Also, JUnit is among the most popular and also among the earliest test frameworks both in industry and academia, thus there is more active research on it. We do not think selection criteria of research papers could be factor for high popularity of JUnit in the results. We did not search explicitly with keyword "JUnit" and the keywords we used in the search (see Table 2) are generic terms and not specific to JUnit. When we examine the Table 2, we see that the type of activity part contains generic software lifecycle terms and the keywords in focus part are generic terms in software testing.

- *RQ 9 (software systems under test):* Studying the type of SUTs used in STCE (RQ 9) reveals that there is a good mixture of SUT types used in the studies: academic experimental systems (or simple code examples), real open-source and commercial systems. The majority of studies (57%) have used open-source systems. Regarding the scale of SUTs, we observe that many studies have used several SUTs, in range of 0–22 SUTs with the average of 3.17. LOC size of the SUTs varies among studies with maximum LOC of 1,459,929. Here we reported the size of SUTs since it is an important external validity threat in empirical studies in testing area and affects the generalizability of the results. Also, results of this study show that ratio of test code size to SUT size is in range of 0.1–1.6. This derived metric shows the scale of automated test suites across different system and is used as a test adequacy measure by some testers, e.g., [121,122]. The implications of these results are as follows: in some studies, considerable amount of test code compared to SUT size itself has been used or developed, e.g., in [54]. However, in some studies such as [2], small amount of test code have been developed for the SUT. Analysis of the SUT size scales showed that various sizes of SUTs have been used in this area.
- *RQ 10 (tools):* Tools (as the most predominant contribution facet) presented for STCE are investigated to find the list of the tools and also their availability for download. Among 40 tools that are proposed for STCE, less than half of the tools (45%) were available for download. It is good to have this percentile of tools to be available, although not perfect, since the availability of tools can lead to higher impact on research community and industry.
- *RQ 11 (academia versus industry and their collaborations):* Data from RQ 11 show that although most of the studies are from academia, joint works between academia and industry have increased strongly in recent years.

The results of this RQ can be used to refine the results of RQ 4. In RQ 4, we concluded that quality assessment and co-maintenance of test-code are two predominant STCE activities in general. While RQ 11 states that this is the case only in academia and not in general. The reason is that most of studies in our study pool, 39 from 60, belonged to academia. Moreover, RQ 11 shows that among the industry practitioners, development and co-maintenance of test-code are more popular. Also, there were considerable amount of joint works in quality assessment. Thus, it can be inferred that quality assessment of test-code is not of less value for industry practitioners, but since it involves more research activities, it needs collaboration of research community with industry practitioners.

7.2. Suggested follow-up research directions for the research community

After aggregating the open issues in the pool of studies, as a result of the above 11 RQs and our SM study, we summarize synthesize, and group below the research directions in the area of STCE, which are worthy of (further) investigation, and can show the future directions of the field to new researchers and PhD students in this area. For each of the proposed research directions, we provide clear mapping with the actual mappings in the primary studies.

7.2.1. Group 1: Semi- and fully-automated development of test-code

Writing high-quality test code is not trivial [37], especially for large-scale software. Although there have been number of studies in this area, there is still need for additional methods, concepts (e.g., test patterns), and tools for development of high quality test code that is easy to verify and also maintain. The following research directions are among those which shall be studied:

- *Research direction 1.1:* What type of test patterns will make test code easy to implement, verify and maintain? This topic was discussed in [27] and several test patterns were presented, but further investigation is needed.
- *Research direction 1.2:* How can we increase the level of automation and reduce manual effort in development of test code? This topic is based on the implications and recommendations of the following studies: [12,18,25,27].

7.2.2. Group 2: Quality assessment and quality improvement of test-code

This objective focuses on both functional and also non-functional quality attributes of test code. The former checks whether a given test suites properly tests the production code under test. Automated test cases can carry faults on their own due to the error-proneness of human activity spent to develop/maintain them. Similar to the quote "Watching the watchman", the quality of automated test scripts should be carefully assessed and assured. If testers develop test suites that are faulty themselves or cannot guard the system under test (SUT) against potential faults, those test suites would be a waste of time and resources.

Again, although there have been a number of studies in this area, there is still need for additional methods, metrics and tools in this field. The following research directions are worthy of further investigation:

- *Research direction 2.1:* What are the quality attributes of test code and why are they important? This research direction is based on the implications of [1–3,5,10].
- *Research direction 2.2:* How can we qualitatively and quantitatively measure the quality of test suites and their impact on production software quality and other project attributes (e.g., effort, cost)? This research direction was suggested in [1–3,5,10].

In follow-up of the above research directions, when quality-related issues are detected, we need to deal with them and improve test-code quality. For example, when parts of a given test code has low maintainability, there is a need to find ways to improve them for the purpose of general software maintainability. Quality improvement activities in this context can also be referred to as "perfective (self-)maintenance" as opposed to co-maintenance with production code. For this purpose, the following directions shall be studied:

- *Research direction 2.3:* How can the quality of test code be improved? This research direction was discussed in [1,5,7,10].
- *Research direction 2.4:* What are the challenges in quality assessment and improvement of test-code? This research direction was explored in [34,36,37,49], however need for further work was mentioned.

7.2.3. Group 3: Co-maintenance and co-evolution of test-code as production code is maintained

For large-scale software, as production (SUT) code is maintained, co-maintenance and co-evolution of test code has been reported to be a challenging and costly task [75,77,123]. As per our literature review, most of the existing studies have studied this phenomenon in exploratory and empirical settings and "after the fact" (e.g., by mining the repositories of open-source software), without offering concrete guidelines and systematic approaches. There is a need for additional work on the latter.

- *Research direction 3.1:* How can co-maintenance and co-evolution of test-code be conducted in a systematic and efficient way, while conserving the quality of test code? This research direction was mentioned as an outcome of RQ 5 in Section 7.1 and also was discussed in [2,4,6,8,9,12].
- *Research direction 3.2:* What are the challenges in co-maintenance and co-evolution of test-code? This research direction was discussed in [38,59,60].

7.2.4. How new researchers can benefit from the above research directions

The aim of SM studies is providing a wide overview of a research area. SM studies help new researchers with an indication of the quantity of the evidence on a research topic [64]. The results of a SM study can be used for identifying areas for more detailed reviews, e.g., SLRs, or to find out areas suitable for primary studies [64]. The research directions that we have suggested above are meant to server as "pointers" and it up to a new researcher to "research and explore" each research direction and develop a research topic based on them. We believe that a novice of the field could start his/her own research by reviewing our SM study and the suggested research directions and then develop his/her own research agenda. For example, based on research direction 1.1, a novice of the field could develop his/her own research agenda from this research direction by studying the existing test patterns in [27] and then explore the type of patterns which are missing and could help increase the quality of test code.

7.3. Potential threats to validity

The main issues related to threats to validity of this SM review are inaccuracy of data extraction, and incomplete set of studies in our pool due to limitation of search terms, selection of academic search engines, and researcher bias with regards to exclusion/inclusion criteria. In the this section, these threats are discussed in the context of the four types of threats to validity based on a standard checklist for validity threats presented in [124].

7.3.1. Internal validity

The systematic approach that has been utilized for article selection is described in Section 4. In order to make sure that this review is repeatable, search engines, search terms and inclusion/exclusion criteria are carefully defined and reported. Problematic issues in selection process are limitation of search terms and search engines, and bias in applying exclusion/inclusion criteria.

Limitation of search terms and search engines can lead to incomplete set of primary sources. Different terms have been used by different authors to point to a similar concept. In order to mitigate risk of finding all relevant studies, formal searching using defined keywords has been done followed by manual search in references of initial pool and in web pages of active researchers in our field of study. For controlling threats due to search engines, not only we have included comprehensive academic databases such as Google Scholar, but also we have searched special active venues related to STCE. Therefore, we believe that adequate and inclusive basis has been collected for this study and if there is any missing publication, the rate will be negligible.

Applying inclusion/exclusion criteria can suffer from researchers' judgment and experience. Personal bias could be introduced during this process. To minimize this type of bias, joint voting is applied in article selection and only articles with high score are selected for this study.

7.3.2. Construct validity

Construct validities are concerned with issues that to what extent the object of study truly represents theory behind the study [124]. Threats related to this type of validity in this study were suitability of RQs and categorization scheme used for the data extraction.

To limit construct threats in this study, GQM approach is used to preserve the tractability between research goal and questions. Research questions are designed to cover our goal and different aspects of STCE. Questions are answered according to a categorization scheme. For designing a good categorization scheme, we have adapted standard classifications from [66] and also have finalized the used schema through several iterative improvement process.

7.3.3. Conclusion validity

Conclusion validity of a SM study provided when correct conclusion reached through rigorous and repeatable treatment. In order to ensure reliability of our treatments, acceptable size of primary sources are selected and terminology in defined schema reviewed by authors to avoid any ambiguity. All primary sources are reviewed by at least two authors to mitigate bias in data extraction. Each disagreement between authors was resolved by consensus among researchers.

Following the systematic approach and described procedure ensured replicability of this study and assured that results of similar study will not have major deviations from our classification decisions.

7.3.4. External validity

External validity is concerned with to what extent the results of our SM study can be generalized. As described in Section 4, defined search terms in the article selection approach resulted in having primary sources all written in English language; studies written in other languages were excluded. The issue lies in whether our selected works can represent all types of literature in the area of STCE. For these issues, we argue that relevant literature we selected in our pool contained sufficient information to represent the knowledge reported by previous researchers or professionals.

Chapter 8 of [124] describes external validity as ability to generalize results to industrial contexts. As it can be seen from the collected data through study, in addition to academic studies, good proportion of industrial and collaborative works exists in our primary sources and acceptable proportion of experiments have done in industrial contexts. This means that our inclusive process of article selection has lead us to have an adequate basis for concluding results useful for academia and applicable in industry.

Also, note that our findings in this study are mainly within the field of STCE. Beyond this field, we had no intention to generalize our results. Therefore, few problems with external validity are worthy of substantial attention.

8. Conclusions and future work

Automated software testing and development of test code are now mainstream in the software industry. Code engineering in the context of software test has been a long standing problem

and many researchers from academia and also practitioners have addressed the challenges in this field. The existence of numerous publications in this field accompanied by the increase in its popularity since 2007, has lead us to conclude that this is an ideal time to start conducting comprehensive secondary studies.

This systematic review investigated the studies in the area of Software Test-Code Engineering (STCE), published between 1999 and 2012 to unveil state-of-the-art in STCE. Through a well-defined article selection process, 60 studies from 72 studies retrieved by our initial search were included in this study. For the purpose of repeatability of our study, the whole selection process is carefully done and reported. The three-fold contributions of this study are a systematic map developed for the area of STCE, Systematic mapping of the existing research in the area of STCE, and providing online article repository which can be used by researchers of this area in their future studies.

For future work, we intend to extend our study and conduct an in-depth Systematic Literature Review (SLR) of the field to identify the benefits and limitation of the different STCE approaches and their effectiveness.

Acknowledgements

Vahid Garousi was supported by Atilim University and the Visiting Scientist Fellowship Program (#2221) of the Scientific and Technological Research Council of Turkey (TÜBİTAK). The Canadian authors were additionally supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the following three Grants: Discovery Grant No. 341511-07, ENGAGE Grant #EGP 444884-12 and ENGAGE Grant #EGP 437020-12.

References

Primary studies

[1] P. Baker, D. Evans, J. Grabowski, H. Neukirchen, B. Zeiss, TRex – the refactoring and metrics tool for TTCN-3 test specifications, in: Proceedings of the Testing: Academic & Industrial Conference on Practice and Research Techniques, 2006, pp. 90–94.

[2] G. Bavota, A. Qusef, R. Oliveto, A.D. Lucia, D. Binkley, An empirical analysis of the distribution of unit test smells and their impact on software maintenance, in: proceedings of the International Conference on Software Maintenance, 2012.

[3] M. Breugelmans, B.V. Rompaey, TestQ: exploring structural and maintenance characteristics of unit test suites, in: Proceedings of the International Workshop on Academic Software Development Tools and Techniques. <http://www.scg.unibe.ch/download/wasdett/wasdett2008-paper07.pdf>, 2008.

[4] J. Chen, M. Lin, K. Yu, B. Shao, When a GUI regression test failed, what should be blamed?, in: Proceedings of the IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 467–470.

[5] W.-K. Chen, J.-C. Wang, Bad smells and refactoring methods for GUI test scripts, in: Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2012, pp. 289–294.

[6] S.R. Choudhary, D. Zhao, H. Versee, A. Orso, WATER: web application TEst repair, in: Proceedings of the First International Workshop on End-to-End Test Script Engineering, 2011, pp. 24–29.

[7] S. Counsell, R.M. Hierons, Refactoring test suites versus test behaviour: a TTCN-3 perspective, in: Fourth International Workshop on Software Quality Assurance: In Conjunction with the 6th ESEC/FSE Joint Meeting, 2007, pp. 31–38.

[8] B. Daniel, T. Gvero, D. Marinov, On test repair using symbolic execution, in: Proceedings of the 19th International Symposium on Software Testing and Analysis, 2010, pp. 207–218.

[9] B. Daniel, V. Jagannath, D. Dig, D. Marinov, ReAssert: suggesting repairs for broken unit tests, in: Proceedings of the 24th IEEE/ACM conference on automated software engineering, 2009, pp. 433–444.

[10] A.v. Deursen, L. Moonen, A.v.d. Bergh, G. Kok, Refactoring test code, in: Extreme Programming Perspectives, 2001, pp. 92–95.

[11] G. Din, D. Vega, I. Schieferdecker, Automated maintainability of ttcn-3 test suites based on guideline checking, in: Proceedings of the 6th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems, 2008, pp. 417–430.

[12] M. Fewster, D. Graham, Software Test Automation: Effective use of Test Execution Tools, Prentice Hall, 1999.

[13] V. Garousi, N. Koochakzadeh, An empirical evaluation to study benefits of visual versus textual test coverage information, in: Proceedings of the 5th International Academic and Industrial Conference on Testing – practice and research techniques, 2010, pp. 189–193.

[14] M. Grechanik, Q. Xie, C. Fu, Experimental assessment of manual versus tool-based maintenance of GUI-directed test scripts, in: Proceedings of the International Conference on Software Maintenance, 2009, pp. 9–18.

[15] M. Grechanik, Q. Xie, C. Fu, Maintaining and evolving GUI-directed test scripts, in: Proceedings of the 31st International Conference on Software Engineering, 2009, pp. 408–418.

[16] M. Greiler, A.v. Deursen, A. Zaidman, Measuring test case similarity to support test suite understanding, in: Proceedings of the 50th International Conference on Objects, Models, Components, Patterns, 2012, pp. 91–107.

[17] E.M. Guerra, C.T. Fernandes, Refactoring test code safely, in: Proceedings of the International Conference on Software Engineering Advances, 2007, p. 44.

[18] P. Gupta, P. Surve, Model based approach to assist test case creation, execution, and maintenance for test automation, in: Proceedings of the First International Workshop on End-to-End Test Script Engineering, 2011, pp. 1–7.

[19] T. Knauth, C. Fetzer, P. Felber, Assertion-driven development: assessing the quality of contracts using meta-mutations, in: Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops, 2009, pp. 182–191.

[20] N. Koochakzadeh, V. Garousi, A tester-assisted methodology for test redundancy detection, J. Adv. Softw. Eng. 2010 (2010) 6:1–6:13 (special issue on software test automation).

[21] N. Koochakzadeh, V. Garousi, TeCReVis: a tool for test coverage and test redundancy visualization, in: Proceedings of the 5th International Academic and Industrial Conference on Testing – Practice and Research Techniques, 2010, pp. 129–136.

[22] N. Koochakzadeh, V. Garousi, F. Maurer, Test redundancy measurement based on coverage information: evaluations and lessons learned, in: Proceedings of the 2009 International Conference on Software Testing Verification and Validation, 2009, pp. 220–229.

[23] K. Koster, D. Kao, State coverage: a structural test adequacy criterion for behavior checking, in: The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers, 2007, pp. 541–544.

[24] F. Lanubile, T. Mallardo, Inspecting automated test code: a preliminary study, in: Proceedings of the 8th International Conference on Agile Processes in Software Engineering and Extreme Programming, 2007, pp. 115–122.

[25] G. Li, I. Ghosh, S.P. Rajan, KLOVER: a symbolic execution and automatic test generation tool for C++ programs, in: Proceedings of the 23rd International Conference on Computer aided Verification, 2011, pp. 609–615.

[26] P. Makedonski, J. Grabowski, H. Neukirchen, Validating the behavioral equivalence of TTCN-3 test cases, in: Proceedings of the First International Conference on Advances in System Testing and Validation Lifecycle, 2009, pp. 117–122.

[27] G. Meszaros, xUnit Test Patterns: Pearson Education. <http://www.xunitpatterns.com>, 2007.

[28] M. Mirzaaghaei, F. Pastore, TestCareAssistant: automatic repair of test case compilation errors, in: Proceedings of the Workshop of the Italian Eclipse Community, 2011.

[29] M. Mirzaaghaei, F. Pastore, M. Pezz, Supporting test suite evolution through test case adaptation, in: Proceedings of the IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 231–240.

[30] M. Mirzaaghaei, F. Pastore, M. Pezze, Automatically repairing test cases for evolving method declarations, in: Proceedings of the IEEE International Conference on Software Maintenance, 2010, pp. 1–5.

[31] N. Nagappan, L. Williams, J. Osborne, M. Vouk, P. Abrahamsson, Providing test quality feedback using static source code and automatic test suite metrics, in: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, 2005, pp. 85–94.

[32] H. Neukirchen, M. Bisanz, Utilising code smells to detect quality problems in TTCN-3 test suites, in: Proceedings of the 19th IFIP TC6/WG6.1 International Conference, and 7th International Conference on Testing of Software and Communicating Systems, 2007, pp. 228–243.

[33] H. Neukirchen, B. Zeiss, J. Grabowski, An approach to quality engineering of TTCN-3 test specifications, Int. J. Softw. Tools Technol. Transfer 10 (2008) 309–326.

[34] H. Neukirchen, B. Zeiss, J. Grabowski, P. Baker, D. Evans, Quality assurance for TTCN-3 test specifications, J. Softw. Test. Verif. Reliab. TAIC Part Special issue 18 (2008) 71–97.

[35] T. Nivas, Test harness and script design principles for automated testing of non-GUI or web based applications, in: Proceedings of the First International Workshop on End-to-End Test Script Engineering, 2011, pp. 30–37.

[36] R. Osherove, The Art of Unit Testing, Manning Publication Co., 2009.

[37] A. Page, K. Johnston, B. Rollison, How We Test Software at Microsoft, Microsoft Press, 2008.

[38] L. S. Pinto, S. Sinha, A. Orso, Understanding myths and realities of test-suite evolution, in: Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering, 2012.

[39] S. Reichhart, T. Gîrba, S. Ducasse, Rule-based assessment of test quality, J. Object Technol. 6 (2007) 231–251.

[40] B.V. Rompaey, B.D. Bois, S. Demeyer, M. Rieger, On the detection of test smells: a metrics-based approach for general fixture and eager test, IEEE Trans. Softw. Eng. 33 (2007) 800–817.
[41] B.V. Rompaey, S. Demeyer, Estimation of test code changes using historical release data, in: Proceedings of the 15th Working Conference on Reverse Engineering, 2008, pp. 269–278.
[42] B. V. Rompaey and S. Demeyer, "Exploring the composition of unit test suites", in Proceedings of the International Conference on Automated Software Engineering 2008, pp. 11–20.
[43] B.V. Rompaey, S. Demeyer, Establishing traceability links between unit test cases and units under test, in: Proceedings of the European Conference on Software Maintenance and Reengineering, 2009, pp. 209–218.
[44] M.J. Rutherford, A.L. Wolf, A case for test-code generation in model-driven systems, in: Proceedings of the 2nd International Conference on Generative Programming and Component Engineering, 2003, pp. 377–396.
[45] D. Schuler, A. Zeller, Assessing oracle quality with checked coverage, in: Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation, 2011, pp. 90–99.
[46] Y. Shewchuk, V. Garousi, Experience with maintenance of a functional GUI test suite using IBM rational functional tester, in: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, 2010, pp. 489–494.
[47] M. Skoglund, P. Runeson, A case study on regression test suite maintenance in system evolution, in: Proceedings of the 20th IEEE International Conference on Software Maintenance, 2004, pp. 438–442.
[48] Y. Song, S. Thummalapenta, T. Xie, UnitPlus: assisting developer testing in eclipse, in: Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange, 2007, pp. 26–30.
[49] S. Thummalapenta, M.R. Marri, T. Xie, N. Tillmann, J.d. Halleux, Retrofitting unit tests for parameterized unit testing, in: Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software, 2011, pp. 294–309.
[50] S. Thummalapenta, S. Sinha, D. Mukherjee, S. Chandra, Automating test automation, in: Proceedings of the International Workshop on End-to-End Test Script Engineering, 2012.
[51] N. Tillmann, J.d. Halleux, Pex–white box test generation for .net, in: Proceedings of the International Conference on Tests and Proofs, 2008, pp. 134–153.
[52] D. Vanoverberghe, J.d. Halleux, N. Tillmann, F. Piessens, State coverage: software validation metrics beyond code coverage, in: Proceedings of the 38th International Conference on Current Trends in Theory and Practice of Computer Science, 2012, pp. 542–553.
[53] D. Vega, G. Din, S. Taranu, I. Schieferdecker, Application of clustering methods for analysing of TTCN-3 test data quality, in: Proceedings of the Third International Conference on Software Engineering Advances, 2008, pp. 237–244.
[54] C. Wiederseiner, V. Garousi, M. Smith, Tool support for automated traceability analysis in embedded software systems, in: Proceedings of the IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011, pp. 1109–1117.
[55] C. Wiederseiner, S.A. Jolly, V. Garousi, M.M. Eskandar, An open-source tool for automated generation of black-box xunit test code and its industrial evaluation, in: Proceedings of the 5th International Academic and Industrial Conference on Testing – Practice and Research Techniques, 2010, pp. 118–128.
[56] Q. Xie, A.M. Memon, Designing and comparing automated test oracles for GUI-based software applications, ACM Trans. Softw. Eng. Methodol. 16 (2007) 1–38.
[57] T. Xie, Augmenting automatically generated unit-test suites with regression oracle checking, in: Proceedings of the 20th European Conference on Object-Oriented Programming, 2006, pp. 380–403.
[58] T. Xie, D. Marinov, D. Notkin, Rostra: a framework for detecting redundant object-oriented unit tests, in: Proceedings of the 19th IEEE International Conference on Automated Software Engineering, 2004, pp. 196–205.
[59] G. Yang, S. Khurshid, M. Kim, Specification-based test repair using a lightweight formal method, in: Proceedings of the 18th International Symposium on Formal Methods, 2012, pp. 455–470.
[60] A. Zaidman, B.V. Rompaey, A.v. Deursen, S. Demeyer, Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining, J. Empirical Softw. Eng. 16 (2011) 325–364.

Other studies

[61] V. Garousi, R. Kotchorek, M. Smith, Test cost-effectiveness and defect density: a case study on the android platform, in: Atif M. Memon (Ed.), Advances in Computers, vol. 89, Elsevier, 2013, pp. 164–205.
[62] Android Team, Android Git repositories. <https://android.googlesource.com> (last accessed 05.12.12).
[63] V. Garousi, T. Varma, A replicated survey of software testing practices in the Canadian province of Alberta: what has changed from 2004 to 2009?, J Syst. Softw. 83 (2010) 2251–2262.
[64] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, in: Evidence-based Software Engineering, 2007.
[65] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering – a systematic literature review, Inf. Softw. Technol. 51 (2009) 7–15.
[66] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Presented at the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), 2008.
[67] J. Jill, M. Lydia, L. Fiona, Doing Your Literature Review: Traditional and Systematic Techniques, SAGE Publications, 2011.
[68] V. Garousi, Online Paper Repository for Software Test-Code Engineering: A Systematic Mapping. <http://www.softqual.ucalgary.ca/projects/SM/STCE> (last accessed 05.12.12).
[69] P. Ammann, J. Offutt, Introduction to Software Testing, Cambridge University Press, 2008.
[70] R.V. Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison-Wesley, New Jersey, 2000.
[71] A.P. Mathur, Foundations of Software Testing, Addison-Wesley Professional, 2008.
[72] P. Tahchiev, F. Leme, V. Massol, G. Gregory, JUnit in Action, Manning Publications Company, 2010.
[73] C.T. Brown, G. Gheorghiu, J. Huggins, An Introduction to Testing Web Applications with Twill and Selenium, O'Reilly Media, 2007.
[74] C. Kaner, J. Bach, B. Pettichord, Lessons Learned in Software Testing, Wiley, 2002.
[75] Z. Lubsen, A. Zaidman, M. Pinzger, Using association rules to study the co-evolution of production & test code, in: Proceedings of the IEEE International Working Conference on Mining Software Repositories, 2009.
[76] A. Qusef, R. Oliveto, A. De Lucia, Recovering traceability links between unit tests and classes under test: an improved method, in: IEEE International Conference on Software Maintenance, 2010, pp. 1–10.
[77] M. Skoglund, P. Runeson, A case study on regression test suite maintenance in system evolution, in: Proceedings of IEEE International Conference on Software Maintenance, 2004, pp. 438–442.
[78] A. Zaidman, B. Van Rompaey, S. Demeyer, A. van Deursen, Mining software repositories to study co-evolution of production & test code, in: International Conference on Software Testing, Verification, and Validation, 2008, pp. 220–229.
[79] E. Hendrickson, The difference between test automation failure and success, in: International Conference on Software Testing, Analysis & Review, 1998.
[80] E. Dustin, T. Garrett, B. Gauf, Why automated software testing fails and pitfalls to avoid, in: Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality, Addison-Wesley Professional, 2009.
[81] D. Graham, M. Fewster, Experiences of Test Automation: Case Studies of Software Test Automation, Addison-Wesley Professional, 2012.
[82] Co-organizer: M. Grechanik, S. Chandra, S. Sinha, The First Workshop International Workshop on End-to-End Test Script Engineering (ETSE), co-located with the International Symposium on Software Testing and Analysis (ISSTA). <http://www.researcher.ibm.com/researcher/view_project.php?id=2188> 2011.
[83] P. Runeson, A survey of unit testing practices, IEEE Softw. 23 (2006) 22–29.
[84] S. Ali, L.C. Briand, H. Hemmati, R.K. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test-case generation, IEEE Trans. Softw. Eng. 36 (2010) 742–762.
[85] F. Elberzhager, J. Münch, V.T.N. Nha, A systematic mapping study on the combination of static and dynamic quality assurance techniques, Inf. Softw. Technol. 54 (2012) 1–15.
[86] V. Garousi, Classification and trend analysis of UML books (1997–2009), J. Softw. Syst. Model. (SoSyM) (2011).
[87] B. Kitchenham, P.B. Keele, D. Budgen, The educational value of mapping studies of software engineering literature, in: Proceedings of ACM/IEEE International Conference on Software Engineering, 2010, pp. 589–598.
[88] W. Afzal, R. Torkar, R. Feldt, A systematic mapping study on non-functional search-based software testing, in: International Conference on Software Engineering and Knowledge Engineering, 2008, pp. 488–493.
[89] M. Palacios, J. García-Fanjul, J. Tuya, Testing in service oriented architectures with dynamic binding: a mapping study, Inf. Softw. Technol. 53 (2011) 171–189.
[90] Z.A. Barmi, A.H. Ebrahimi, R. Feldt, Alignment of requirements specification and testing: A systematic mapping study, in: Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 476–485.
[91] P.A.d.M.S. Neto, I.d.C. Machado, J.D. McGregord, E.S.d. Almeida, S.R.d.L. Meira, A systematic mapping study of software product lines testing, Inf. Softw. Technol. 53 (2011) 407–423.
[92] E. Engström, P. Runeson, Software product line testing – a systematic mapping study, J. Inf. Softw. Technol. 53 (2011) 2–13.
[93] C.R.L. Neto, P.A.d.M.S. Neto, E.S.d. Almeida, S.R.d.L. Meira, A mapping study on software product lines testing tools, in: Proceedings of International Conference on Software Engineering and Knowledge Engineering, 2012.
[94] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Inf. Softw. Technol. 51 (2009) 957–976.
[95] Z. Zakaria, R. Atan, A.A.A. Ghani, N.F.M. Sani, Unit testing approaches for BPEL: a systematic review, in: Proceedings of the Asia–Pacific Software Engineering Conference, 2009, pp. 316–322.
[96] A.T. Endo, A.d.S. Simao, A systematic review on formal testing approaches for web services, in: Brazilian Workshop on Systematic and Automated Software Testing, International Conference on Testing Software and Systems, 2010, p. 89.

[97] S. Ali, L.C. Briand, H. Hemmati, R.K. Panesar-Walawege, A systematic review of the application and empirical investigation of search based test case generation, IEEE Trans. Softw. Eng. 36 (2010) 742–762.
[98] E. Engström, P. Runeson, M. Skoglund, A systematic review on regression test selection techniques, J. Inf. Softw. Technol. 53 (2010) 14–30.
[99] R.V. Binder, Testing object-oriented software: a survey, in: Proceedings of the Tools-23: Technology of Object-oriented Languages and Systems, 1996, p. 374.
[100] N. Juristo, A.M. Moreno, S. Vegas, Reviewing 25 years of testing technique experiments, Empirical Softw. Eng. 9 (2004) 7–44.
[101] P. McMinn, Search-based software test data generation: a survey: Research articles, Softw. Test. Verif. Reliab. 14 (2004) 105–156.
[102] M. Grindal, J. Offutt, S.F. Andler, Combination testing strategies: a survey, Softw. Test. Verif. Reliab. 15 (2005).
[103] G. Canfora, M.D. Penta, Service-oriented architectures testing: a survey, in: International Summer Schools on Software Engineering, 2008, pp. 78–105.
[104] C.S. Păsăreanu, W. Visser, A survey of new trends in symbolic execution for software testing and analysis, Int. J. Softw. Tools Technol. Transfer 11 (2009) 339–353.
[105] M. Bozkurt, M. Harman, Y. Hassoun, Testing Web Services: A Survey, Technical Report TR-10-01, Department of Computer Science, King's College London, 2010.
[106] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Softw. Eng. 37 (2011) 649–678.
[107] P.A.d.M.S. Neto, P. Runeson, I.d.C. Machado, E.S.d. Almeida, S.R.d.L. Meira, E. Engström, Testing software product lines, IEEE Softw. 28 (2011) 16–20.
[108] A.M. Memon, B.N. Nguyen, Advances in automated model-based system testing of software applications with a GUI front-end, Adv. Comput. 80 (2010) 121–162.
[109] T.D. Hellmann, A. Hosseini-Khayat, F. Maurer, Agile interaction design and test-driven development of user interfaces – a literature review, in: Agile Software Development: Current Research and Future Directions, 2010.
[110] V. Garousi, A. Mesbah, A. Betin-Can, S. Mirshokraie, A systematic mapping study of web application testing, Inf. Softw. Technol. 55 (8) (2013) 1374–1396.
[111] I. Banerjee, B. Nguyen, V. Garousi, A. Memon, Graphical user interface (GUI) testing: systematic mapping and repository, Inf. Softw. Technol. 55 (10) (2013) 1679–1694.
[112] I. Singh, A Mapping Study of Automation Support Tools for Unit Testing, Mälardalen University, School of Innovation, Design and Engineering, Sweden – Västerås, 2012.
[113] S. Wang, J. Offutt, Comparison of unit-level automated test generation tools, in: Fifth Workshop on Mutation Analysis, 2009, pp. 2–10.
[114] V.R. Basili, Software Modeling and Measurement: The Goal/Question/Metric paradigm, Technical Report, University of Maryland at College Park, 1992.
[115] N. Bencomo, S. Hallsteinsen, E. Santana de Almeida, A view of the dynamic software product line landscape, IEEE Comput. 45 (2012) 36–41.
[116] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a survey, Softw. Test. Verif. Reliab. 22 (2010) 67–120.
[117] M.H. Alalfi, J.R. Cordy, T.R. Dean, DWASTIC: automating coverage metrics for dynamic web applications, in: Proceedings of the ACM Symposium on Applied Computing, 2009.
[118] T. Dao, E. Shibayama, Coverage criteria for automatic security testing of web applications, Inf. Syst. Secur. 6503 (2011) 111–124 (Lecture Notes in Computer Science).
[119] M. Grechanik, A journey of test scripts: from manual to adaptive and beyond, in: International Workshop on End-to-End Test Script Engineering, 2012.
[120] M. Harder, B. Morse, M.D. Ernst, Specification Coverage as a Measure of Test Suite Quality, September 25, 2001.
[121] Cunningham & Cunningham Inc., Production Code Vs Unit Tests Ratio. <http://www.c2.com/cgi/wiki?ProductionCodeVsUnitTestsRatio> (last accessed December 2013).
[122] T. Bhat, N. Nagappan, Evaluating the efficacy of test-driven development: industrial case studies, in: Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering, 2006, pp. 356–363.
[123] A. Zaidman, B.V. Rompaey, A.v. Deursen, S. Demeyer, Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining, Empirical Softw. Eng. 16 (2011) 325–364.
[124] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, 2000.
[125] S. Doğan, A. Betin-Can, V. Garousi, Web application testing: a systematic literature, J. Syst. Softw. 91 (2014) 174–201.