Information and Software Technology 54 (2012) 1092–1106

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

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

# Reducing test effort: A systematic mapping study on existing approaches

Frank Elberzhager [a,*], Alla Rosbach [a], Jürgen Münch [b], Robert Eschbach [c]

[a] Fraunhofer Institute for Experimental Software Engineering (IESE), Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
[b] P.O. Box 68 (Gustaf Hällströmin katu 2b), 00014 Helsinki, Finland
[c] ITK Engineering AG, Luitpoldstraße 59, 76863 Herxheim, Germany

ARTICLE INFO

*Article history:*
Received 16 August 2011
Received in revised form 24 April 2012
Accepted 28 April 2012
Available online 18 May 2012

*Keywords:*
Efficiency improvement
Mapping study
Quality assurance
Software testing
Test effort reduction

ABSTRACT

*Context:* Quality assurance effort, especially testing effort, is often a major cost factor during software development, which sometimes consumes more than 50% of the overall development effort. Consequently, one major goal is often to reduce testing effort.
*Objective:* The main goal of the systematic mapping study is the identification of existing approaches that are able to reduce testing effort. Therefore, an overview should be presented both for researchers and practitioners in order to identify, on the one hand, future research directions and, on the other hand, potential for improvements in practical environments.
*Method:* Two researchers performed a systematic mapping study, focusing on four databases with an initial result set of 4020 articles.
*Results:* In total, we selected and categorized 144 articles. Five different areas were identified that exploit different ways to reduce testing effort: approaches that predict defect-prone parts or defect content, automation, test input reduction approaches, quality assurance techniques applied before testing, and test strategy approaches.
*Conclusion:* The results reflect an increased interest in this topic in recent years. A lot of different approaches have been developed, refined, and evaluated in different environments. The highest attention was found with respect to automation and prediction approaches. In addition, some input reduction approaches were found. However, in terms of combining early quality assurance activities with testing to reduce test effort, only a small number of approaches were found. Due to the continuous challenge of reducing test effort, future research in this area is expected.

## 1. Introduction

High-quality software is of crucial importance today. However, ensuring high-quality software is often costly. Several studies have shown that one major cost factor is inadequate quality assurance, especially inadequate testing. Late fault and failure detection normally leads to huge correction and maintenance costs. Quality assurance efforts, especially testing efforts, often consume more than 50% of the overall development efforts [121,136]. Reducing test effort by accepting quality reductions is often not acceptable. Therefore, an open problem is how to reduce test effort without forgoing the quality level of the final software. Unfortunately, an overview of existing approaches is missing and little guidance is available regarding the selection of appropriate approaches.

The main objective of this systematic mapping study is the identification of existing approaches to testing effort reduction. The results of this study are expected to support practitioners by identifying options with respect to improving their quality assurance strategies. Researchers might benefit by getting an overview of current approaches and indications of existing evidence regarding their use, as well as by identifying research gaps and future research directions.

Several existing systematic mapping studies and systematic literature reviews focusing on related topics already exist. Kitchenham et al. [156] and Da Silva et al. [155] identified and summarized a large number of software engineering literature surveys. Based on their results, the following surveys can be seen as closely related to the mapping study presented in this article: Juristo et al. [157] and Runeson et al. [158] surveyed and analyzed existing literature on quality assurance techniques and the empirical knowledge gathered with such techniques. In particular, testing and inspection techniques were analyzed. However, effort reduction was not particularly considered. Zakaria et al. [159] performed a study surveying and analyzing literature on unit test techniques for BPEL. Several research questions were addressed, such as the applicability of testing techniques in a web service environment, empirical evidence, and efficiency aspects. The focus in that study

* Corresponding author. Tel.: +49 631 6800 2248; fax: +49 631 6800 92248.
*E-mail addresses:* frank.elberzhager@iese.fraunhofer.de (F. Elberzhager), alla.rosbach@iese.fraunhofer.de (A. Rosbach), juergen.muench@cs.helsinki.fi (J. Münch), robert.eschbach@itk-engineering.de (R. Eschbach).

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.04.007

is not explicitly on effort reductions. More literature surveys exist that focus on specific aspects such as fault prediction models [160] or test automation [161]. Those studies cover a large number of existing publications. Therefore, they present an appropriate overview of the relevant literature regarding the specific topics. These surveys are expected to be helpful for identifying the relevant literature on the specific topic and can be seen as additional sources of information. However, a broader overview of approaches that allows for test effort reduction is missing.

We analyzed more than 4020 research articles in our systematic mapping study. In the end, 144 articles were selected and used for classifying existing approaches. We identified five different areas that exploit different ways to reduce testing effort: approaches that predict defect-prone parts or defect content, automation, test input reduction approaches, quality assurance techniques applied before testing, and test strategy approaches. About three fourths of the articles present evaluation results, but details about the context are often missing. Furthermore, results are often preliminary, unspecific, or even contradictory.

This paper is organized as follows: Section 2 provides the method used to perform this mapping study. Furthermore, it describes the research questions, the search string used, and seven phases applied for choosing articles. In Section 3, the main results of this mapping study are presented and threats to validity are outlined. A discussion and implications are given in Section 4. Section 5 presents the final conclusions that could be drawn from this mapping study together with future work.

## 2. Research methodology

The objective of this systematic mapping study is to give an overview of the state of the art regarding existing approaches that are able to reduce the effort when applying testing techniques. A second goal is to identify gaps and future research topics. We performed this systematic mapping study according to Petersen et al. [110], and enhanced our procedure by some concepts used in a systematic literature review [75] (e.g., using a protocol). In the first step, the research questions were defined. Next, we selected suitable reference databases and identified appropriate search strings. The articles found were excluded or included in order to obtain a final result set based on the defined inclusion and exclusion criteria. Finally, the relevant data was extracted and synthesized for the presentation of the main findings.

### 2.1. Research questions

The overall objective of the systematic mapping study is to identify approaches that are able to reduce testing effort. Reducing effort when applying testing techniques is one way of improving the efficiency of testing. Efficiency can be defined as the number of defects found within a certain time interval, e.g., the number of defects found per hour. Consequently, improving efficiency means finding at least the same number of defects within less time in the scope of this mapping study. The overall goal is divided into four detailed research questions:

RQ1. What are existing approaches for reducing effort when applying testing techniques, and how can they be classified?
RQ2. Which concrete techniques exist to reduce testing effort?
RQ3. How many existing optimization approaches had been evaluated and how had they been evaluated?
RQ4. When were existing optimization approaches published and which publication channels were used?

### 2.2. Study search strategy

Identifying the selection strategy is the first step towards a successful identification of primary studies. The goal of this step of the systematic mapping study is to ensure that the selected articles are complete to the extent possible. It contains two steps: (i) selection of the reference databases and identification of the search strings and (ii) inclusion or exclusion of articles based on the defined inclusion and exclusion criteria. The systematic mapping study was performed by two researchers. Both of them are mainly working in the area of software inspections and software testing.

#### 2.2.1. Source selection and search string

During a systematic mapping study, reference databases are searched with appropriate search strings. The result is a set of articles. These articles are primarily used for answering the research questions. For this reason, a precise determination of the relevant reference databases and the appropriate search strings is necessary. The decision was made to use the following four reference databases:

- Inspec.
- Compendex.
- IEEE Xplore.
- ACM Digital Library.

The main reason for choosing these libraries was that they constitute some of the most relevant sources in software engineering. Inspec and Compendex are comprehensive databases containing millions of publications, especially in the engineering and computer science domain. Moreover, these two databases are accessed by using the Engineering Village interface, which is considered user-friendly and provides advanced search features. IEEE Xplore and the ACM Digital Library are not covered by the first two online libraries. Therefore, they were used in addition to complete the final result set.

The search string was formulated as follows: First, the main search keywords were defined based on our previously defined research questions. We considered terms such as quality assurance, effort, and reduction. Next, a set of relevant synonyms for the main search keywords was identified (e.g., supported by the IEEE Standard Glossary of Software Engineering Terminology [138]). In the last step, the entire search string was generated. The main keywords were connected with the chosen synonyms using the logical operators AND and OR. The search string used for selecting articles from the databases is shown in Fig. 1.

The entire search string was used to search in the reference databases Compendex and Inspec. The database fields title, abstract, and keywords were considered. The entire search string could not be applied in IEEE Xplore and the ACM Digital Library due to its length (e.g., IEEE Xplore only allows five wildcards [80]). Instead, a simplified search string was used. The search string

[("quality assurance" or verification or testing or test or inspection or review)
AND
software
AND
(effort or "test* time" or "time of test*" or "cost of test*" or "test* cost")
AND
(reduc* or minimi* or decreas* or prioriti* or focus or select* or allocation)]

**Fig. 1.** Search string.

included the main keywords such as quality assurance, testing effort, time reduction, and cost reduction.

### 2.2.2. Study selection based on inclusion and exclusion criteria

We formulated a set of suitable inclusion and exclusion criteria for selecting relevant articles. If a certain article discussed the defined overall research question, it was included in the set of selected articles. Otherwise, if a certain article complied with the exclusion criteria, it was discarded. In cases where one team member was not sure whether to include or exclude an article, both researchers discussed the article in a meeting and a decision was made together.

The following inclusion criteria were applied: We included articles that discussed different test optimization techniques and approaches that could help to reduce the testing effort. Test optimization can be supported by combinatorial software testing [55,85] or by white-box approaches [6], for example. However, we are aware that some techniques are able to improve effectiveness (i.e., find more defects) and reduce testing effort at the same time. In this case, the corresponding article was also included even if a broader scope is addressed. Furthermore, techniques or tools used before testing that reduce the number of defects were also included. Finally, only articles written in English were included.

To ensure that the most relevant articles were found, we set the starting year to 1991. Articles published before 1991 were considered as irrelevant and their implication for today's effort reduction was considered as low. Articles within the context of improving testing but with an emphasis on effectiveness were excluded. Furthermore, articles that included only a description of test techniques without concentrating on effort reduction aspects and articles that introduced complete proceedings were also considered as irrelevant.

The two team members conducted the selection of articles for the study in the middle of 2010. The main phase of this selection comprised exclusion based on titles, abstracts, and full text. In carrying out the six selection phases, we applied our defined inclusion and exclusion criteria. A seventh phase was added in 2011 for complementing the main findings with additional articles published in 2010 that could not have been found during the first selection procedure.

*2.2.2.1. Phase 0.* The search string (see Fig. 1) was used to search in the two databases Inspec and Compendex. We found 4020 articles in total. In order to manage such a large number of articles, we used the open source reference tool Zotero [135]. This tool allows adding notes, tags, and attachments to each article. The 4020 found articles were imported into Zotero and sorted by titles. After manual elimination of duplicates, 3150 articles remained.

*2.2.2.2. Phase 1.* Prior to the inclusion and exclusion of articles based on the titles, the common understanding of the two team members regarding the selection of articles was measured. For this purpose, the Cohen's Kappa value for measuring agreement between two members was calculated. Each researcher independently assessed 110 articles. We rated most of the articles equally (91 articles). Nineteen articles were rated differently. The inclusion or exclusion of these articles was discussed afterwards. The resulting Cohen's Kappa value was $K = 0.71$. This result shows that the two team members exhibited a substantial degree of agreement between them. Based on this rating, some exclusion and inclusion criteria were clarified again.

*2.2.2.3. Phase 2.* Next, we conducted the inclusion based on titles. In order to reduce study selection effort, the 3150 articles were divided into two groups. Each team member had to independently check one set of articles and decide whether the articles should be included or excluded based on the titles. Included articles, excluded articles, and articles classified as 'not sure' were recorded in a protocol. If a team member was not sure whether to include or exclude an article, the article was discussed later in a meeting with the other team member and a decision was made. At first, 273 articles were included, 2780 articles were excluded, and 97 articles were classified as not sure. The main exclusion criteria were complete proceedings (as those do not provide articles directly, but only the title of a conference and sometimes a general introduction), articles from different domains and areas (e.g., focusing on hardware testing), and articles that do not consider any improvement (e.g., focusing only on allocation of effort). In a later meeting, the two team members discussed the 97 articles which were classified as not sure. Of the 97 articles, 29 articles were included and 68 articles were excluded in the end. The main result of this phase is that a total of 302 relevant articles were identified based on titles.

*2.2.2.4. Phase 3.* In this phase, the abstracts of 302 articles were read. Similarly as before, the articles were divided into two sets, one for each team member. The researchers independently read the abstracts of the appropriate articles and again classified the articles as 'include', 'exclude', or 'not sure'. Before the independent selection was done, 20 abstracts were read by both researchers and the consistency with the decisions was checked. Based on those abstracts, the decision was changed for only three papers. 167 were included, 100 were excluded, and 35 articles were classified as 'not sure'. At the next meeting, the two team members decided that nine articles from the latter group should be included in the study. Consequently, the number of relevant articles was 176 after this phase. The main exclusion criteria in this phase were again articles from different domains as well as articles that do not consider any improvement, but describe only a certain test technique.

*2.2.2.5. Phase 4.* In the next step of the study selection, we decided whether to include or exclude a certain article based on the full text of the article. The relevant articles were also divided into two sets as in the previous phases. They were checked independently and finalized in a meeting. Before the independent selection was done, 20 full texts were read by both researchers and the consistency with the decisions was checked again. Based on those articles, the decision was changed for only one paper. In total, 176 articles were investigated, of which 121 were included and 55 were excluded. The main exclusion criteria were articles that compare different testing techniques with another focus (e.g., the user friendliness of various testing techniques) or describe a single testing technique without focusing on effort improvements.

While carrying out the study selection based on the articles found by Inspec and Compendex, we identified 121 articles that were relevant. In the next step, the databases IEEE Xplore and ACM Digital Library had to be searched to ensure that we did not miss any relevant article.

*2.2.2.6. Phase 5.* It was not possible to apply the original search string in IEEE Xplore and the ACM Digital Library since these databases have a simplified search user interface. For this reason, the search string was simplified. The new search string included only the main keywords, e.g., quality assurance, testing, reduction, test effort, and software. The databases IEEE Xplore and ACM Digital Library were searched with the new search string. The study selection was carried out in the same manner as described above for Inspec and Compendex. While performing the study selection, we found seven relevant articles in addition, which were not included in the study selection from Inspec and Compendex. It should be mentioned that the most relevant articles from IEEE Xplore and ACM Digital Library had already been found in Inspec

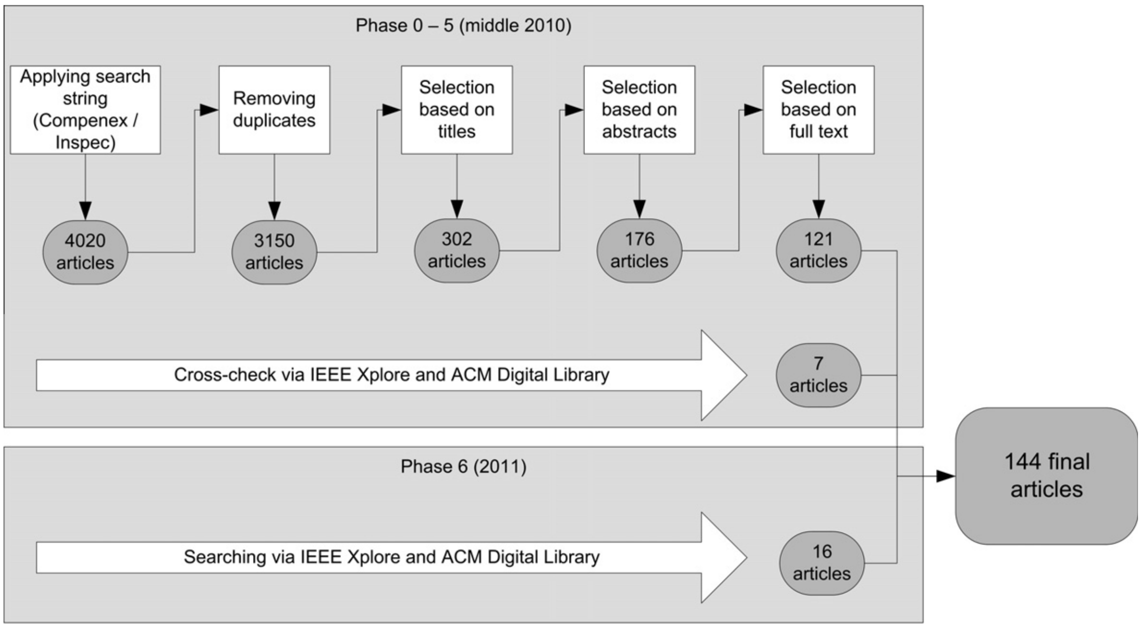
Phase 0 – 5 (middle 2010)
Applying search string (Compenex / Inspec)
Removing duplicates
Selection based on titles
Selection based on abstracts
Selection based on full text
4020 articles
3150 articles
302 articles
176 articles
121 articles
Cross-check via IEEE Xplore and ACM Digital Library
7 articles
Phase 6 (2011)
Searching via IEEE Xplore and ACM Digital Library
16 articles
144 final articles

**Fig. 2.** Overview of the search procedure.

and Compendex. Thus, the main work was done while carrying out the study selection from these databases.

Finally, 121 relevant articles were selected from the databases Inspec and Compendex and seven articles from the databases IEEE Xplore and ACM Digital Library.

*2.2.2.7. Phase 6.* As the analysis and the dissemination of the results needed some time, we decided to scan again for articles that were published in 2010 after the initial search (see phases 0–5). Based on the experience made with Compendex and Inspec and the fact that about 90% of the articles were considered as irrelevant, we focused only on IEEE Xplore and ACM Digital Library in this phase. Those databases cover a suitable number of publications, and we had already found many of the selected articles using the other two databases before. This phase was only conducted by one of the two researchers since the other one was unavailable. Sixteen new articles were found.

In total, 144 relevant articles were found. Fig. 2 presents an overview of the search procedure and the number of articles we selected after each phase.

### 2.3. Data extraction and synthesis

We performed the data extraction according to the procedure described in the previous section. Microsoft Excel was used to summarize the relevant data. During data extraction, we analyzed the full texts of all 144 primary papers. The relevant articles were divided into two sets, one set for each team member. Then each team member read the assigned articles and extracted the required data from the articles in order to answer the corresponding research questions. We collected three kinds of information from each paper. In the first step, standard information was gathered, like title of the article, author's name, the whole reference, the publication channel, and the type of publication (e.g., conference, symposium, journal).

The second and third steps included information directly related to answering the previously defined research questions. In the second step, the goal was to identify for each relevant article the kinds of approaches and methods used to reduce testing effort. At the same time, a first rough classification of those approaches and methods was made, supported by a keywording approach [110]. The final step included gathering information about the evaluation of the approaches and methods. Based on different scoring rubrics stated by Ivarsson and Gorschek [137], we considered some of those aspects in our analysis. Our objective was to identify whether the existing optimization approaches had been evaluated and how they had been evaluated. This means we extracted information on the kind of evaluation that had been performed (i.e., research method, such as lessons learned, case study) and on the context in which the evaluation had been performed (e.g., industrial or academic). The results of the synthesized data are presented in the next section.

## 3. Results

The results of the systematic mapping study are presented next, ordered along the four research questions.

### 3.1. RQ1: What are existing approaches for reducing effort when applying testing techniques, and how can they be classified?

The purpose of research question one was to investigate which approaches for reducing testing effort when applying testing techniques exist and how they can be classified. Petersen et al. [110] suggest using keywording to develop a classification. Following this approach, abstracts are considered first for extracting relevant keywords, which can be complemented by adding keywords from introductions and conclusions. We also followed this approach when extracting the main ideas in each article about how testing effort could be reduced, and clustered the different approaches afterwards. Table 1 first presents the five main categories that were identified. Furthermore, the table shows the distribution of articles based on the different categories of the classification.

Most articles, namely about 50%, discuss test automation. This result is not surprising, because when applying test automation approaches, a lot of testing time can be saved. One example is automating manual effort-consuming tasks such as automated test execution instead of manual performance, automated test report generation instead of manual derivation, or analysis of the test results instead of manual analysis of a huge set of data. Test tools often support automation. A refinement of this category can be

**Table 1**
Distribution of articles based on classification.

| Category | # Articles | % |
|---|---|---|
| Test automation | 71 | 49.3 |
| Prediction | 41 | 28.5 |
| Test input reduction | 22 | 15.2 |
| Quality assurance (QA) before testing | 7 | 4.9 |
| Test strategy | 3 | 2.0 |
| | 144 | 100% |

made with respect to different test phases, such as planning, preparation, execution, and analysis.

Predictions are the next major kind of approaches that can be used to reduce testing effort. About 28% of the articles fall into this category. Predictions can support decisions on how much testing effort is needed or how testing effort should be distributed. For example, data about defect-proneness of modules can be used to focus testing effort. A distinction into prediction approaches for defect content and defect-proneness was made. Defect content prediction means how many defects are expected; defect-proneness prediction means in which areas defects are expected.

The category test input reduction contains about 15% of all articles. This category includes mainly articles that propose different test case selection and prioritization approaches. They are summarized as test suite reduction approaches. In addition, a test sequence reduction approach was found. Finally, comparison studies have evaluated different test input reduction approaches.

Running test cases in order to test a software system usually requires a lot of time. At first, appropriate test data and the expected output have to be determined for each test case. Then, all test cases have to be executed. Finally, the results of the test case execution have to be evaluated. Selecting appropriate test cases and excluding redundant test cases can save a lot of testing time and thus, improve the efficiency of testing.

Test input reduction approaches can be applied in the area of regression testing, whose aim is to test a modified software system. To rerun all of the test cases in the original test suite is quite expensive and takes a lot of testing time. Thus, it is necessary to optimize the test suite. For this reason, different regression test selection methods exist that reuse test cases from an original test suite.

Quality assurance before testing (QA before testing) was only found in about 5% of the articles. In these articles, results from static quality assurance activities, such a code inspection and reviews, are used to reduce effort for later testing activities. Defects already found by inspections are generally less costly than defects found by testing activities.

Furthermore, results from inspections that are performed before testing can be used to make a decision on how much testing effort is needed as a minimum, or how to distribute available testing effort.

The category test strategy includes articles within the context of test optimization that address comprehensive approaches, such as selecting different test techniques for various test levels to save test time (and be more effective).

We performed an automated analysis creating word clouds for all 144 selected articles and for each category. Considering all articles, the most prominent words were test or testing and software, which represent our main context. Furthermore, some prominent word groups containing similar terms were found that show particular directions in this area, such as the following:

- *Data, number, values, results, metrics, study:* A lot of data is given, analyzed, or created to substantiate approaches.
- *Execution, analysis, generation:* Typical test phases are considered.
- *Model, models, algorithm, method, techniques:* A lot of different approaches are used to address testing.
- *Time, cost, effort:* Those terms are relevant for addressing our main goal.
- *Fault, faults, defect:* The main goal of testing as such is finding problems.
- Suite, cases, set: Test suites, test cases, and test sets are often considered and approaches are aimed at reducing them.

With respect to the word clouds for each category, we found prominent words that express the categories quite well, which is an indicator for a suitable categorization of the articles found. For example, each test automation sub-category describes what is the main focus of the articles, such as generation of tests (typical terms here are generation, coverage, path, or input) or analyzing the results (typical terms here are results, information, data, or faults). An overview of the most prominent terms for the different categories can be found in Table 2.

### *3.2. RQ2: Which concrete techniques exist to reduce testing effort?*

Next, we will explain each of the five categories in detail regarding how testing effort can be saved with respect to the identified techniques and approaches.

#### *3.2.1. Category automation*

The category automation with 71 articles contains most of the articles found in this systematic mapping study. The result is not surprising because applying test automation approaches and using test tools offers many advantages, e.g., saving testing effort due to reduced execution time. Manual intervention for repeated tasks and effort to run tests manually can also be reduced.

In this study, we use typical phases of a test process to categorize the large number of test automation approaches.

A test process for software usually includes the following phases: planning, preparation, execution, and analysis. In the planning phase, a test plan for the activities that will be carried out during testing has to be defined. In particular, a test plan defines the test goals and indicates what should be tested, how it should be tested, and by whom the test should be executed.

**Table 2**
Word cloud analysis for each category.

| Category | Prominent terms |
|---|---|
| Test automation (preparation) | Test, cases, generation, data, coverage, program, path, input, method, number, software |
| Test automation (execution) | Test, case, software, component, data, execution, method, numbers, segments, system, tool |
| Test automation (analysis) | Test, case, generated, changes, data, results, information, version, value, statistical, coverage, behavioral, number, fault |
| Prediction (defect content) | Software, testing, faults, model, reliability, number, optimal, testing-effort, system, cost |
| Prediction (defect-proneness) | Software, testing, metrics, prediction, defect, data, faults, complexity, files, system, model, class, code |
| Test input reduction | Test, cases, coverage, selection, reduction, regression, suite, modified, algorithm, fault, program |
| QA before testing | Software, inspection, defects, faults, testing, code, detection, tools, techniques |
| Test strategy | Test, levels, cases, techniques, system, technologies, reuse, software, strategy, faults, effort, phases |

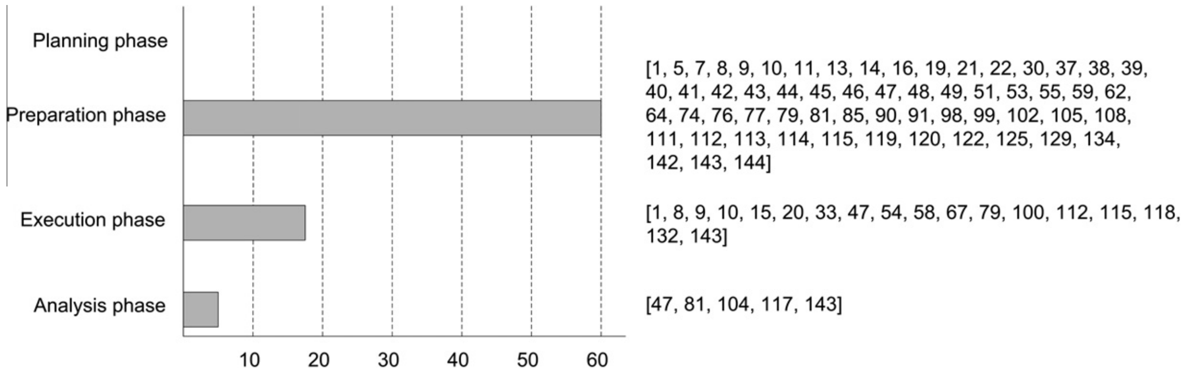

**Fig. 3.** Distribution of articles in the automation category. (See above-mentioned references for further information.)

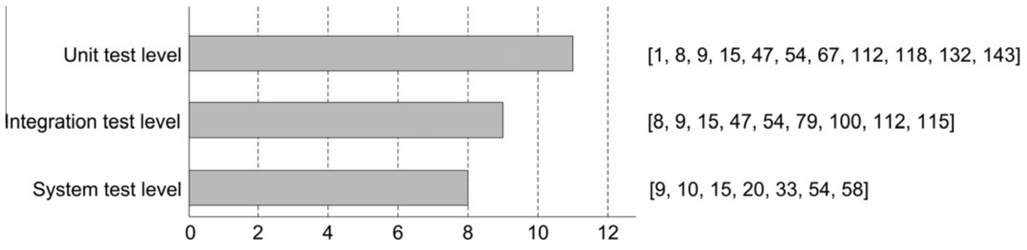

**Fig. 4.** Distribution of articles in execution phase. (See above-mentioned references for further information.)

In the preparation phase, test scenarios, test cases, test data sets, and test scripts needed for test execution have to be developed. During test execution, testers execute the developed software (or parts of it). This phase is based on the test plan defined before. After the test results are reported, a decision should be made in the analysis phase on whether the test is sufficient or whether further tests are necessary. Some of the articles could be classified into more than one phase (see Fig. 3).

First of all, no articles were found that can be classified into the planning phase.

Second, 60 articles were found that discuss the preparation phase. More specifically, different methods, approaches, and techniques for test case generation were described in these articles. Most techniques are supported by tools for automating an entire test process or just certain parts of test processes. For example, automated test case definition substitutes or extends manual test case derivation and is thus much faster. Consequently, effort can be saved. In addition, a lot more test cases can be defined with automation. Various coverage criteria are considered during test case definition, such as branch coverage, statement coverage, data flow coverage, and mc/dc coverage.

We observed that articles use different terms to describe test case generation (e.g., test case generation, test data generation, or test suite generation). However, they mainly perform the same activities. Thus, we categorized these articles in one group.

In most articles, concrete test case generation techniques and methods are proposed. Just a few articles present detailed case studies. For example, Bertolino et al. [1] conducted a case study that aimed at improving the branch testing process. They validated a method for the automatic generation of test paths, and defined a bound on the number of tests needed.

The generation of test cases is based on different algorithms and methods. Again, one frequently pursued goal here is to reduce the effort for defining test cases. In some articles, information about the test basis is presented, e.g., using specifications and UML diagrams for the generation of test cases.

Furthermore, there are some articles that present special techniques and methods for improving the test case generation process. For example, Li et al. [53] propose a method for guiding users through test case generation. An experiment of this method showed a good result regarding test effort reduction.

18 articles were assigned to the execution phase in the testing process. These articles mainly propose different techniques, methods, and approaches for reducing the execution time of test cases. To support the proposed techniques, several testing tools were developed that automate test execution on the following test levels: unit, integration, and system test level (see Fig. 4). Some of these articles discuss various test levels.

Only five articles were found that could be assigned to the analysis phase [47,81,104,117,143]. Gerlich et al. [81] propose an approach that automatically generates test cases based on information about the source code (prototype specification and code structure). Furthermore, this approach includes automated test evaluation and comprehensive presentation of results. The method described in [47] allows performing conformance evaluation considering different levels. Xie et al. [104] present an approach that focuses on analyzing and maintaining test scripts in order to be more efficient. Travison and Staneff [117] present a pattern that helps to automate some steps of the manual process of failure identification by using instrumentation, annotation, and recognition techniques. Finally, Jin et al. [143] analyze outputs of test cases from different versions for focusing testing. Due to the automation, a reduction of the effort can be assumed.

### 3.2.2. Category prediction

The category prediction includes 41 articles (see Fig. 5). In this category, two main sub-categories are distinguished: prediction of the number of expected defects and prediction of the defect-prone parts of a system. Predicting the number of defects can make it easier to decide when to stop testing; predicting defect-prone parts of the system can focus testing efforts. A large variety of concrete methods was found. Many of these approaches present empirical evaluations.

One article discusses how to decide when to stop testing based on the cost-estimation models COCOMO2 and COQUALMO [78]. For this purpose, consideration is primarily given to risk. Thirteen

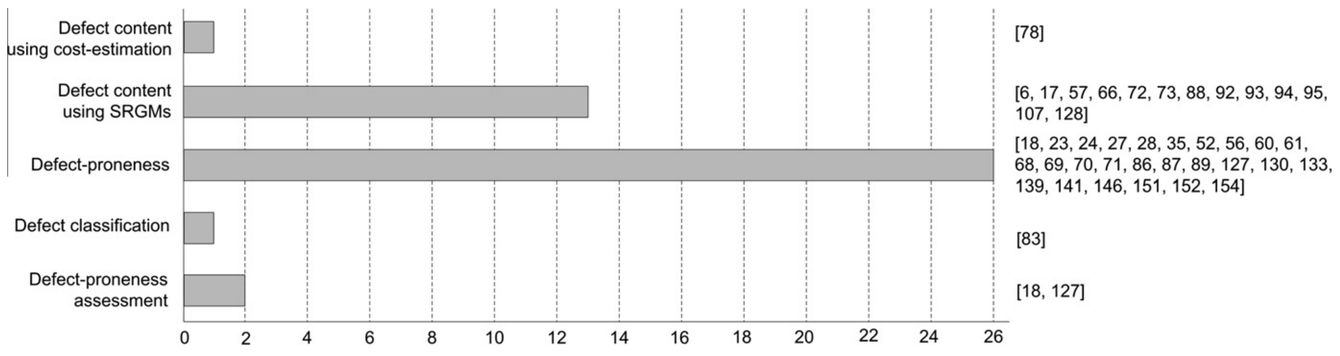

**Fig. 5.** Distribution of articles in category prediction. (See above-mentioned references for further information.)

articles discuss the prediction of software reliability and the remaining number of expected defects. These approaches are based on software reliability growth models (SRGMs).

SRGMs attempt to predict software reliability using test data. The test data are collected during the test execution phase. The SRGMs try to correlate found failure data with known mathematical functions such as an exponential function. If there is a good correlation, the known function can be used to estimate the reliability of the software system to be developed and to predict the number of defects remaining in the software. This knowledge can help to make a decision on whether or not the software is ready for release and how much more testing is required if the software is not ready for release. In particular, a decision such as when to stop testing can be made, which helps to reduce the effort of testing activities. The SRGMs can also be used to estimate how many failures will occur when the software system is used in the field.

There are various types of SRGMs. All these models are based on different assumptions, for instance, experience of the testers; defects are repaired immediately as soon as they are detected; no new code is implemented during the testing phase; failures have different failure rates and failures with the highest rates should be corrected first.

Fig. 6 gives an overview of the aspects of the software development and testing process that the models focus on in particular. The article by Htoon and Thein [88] has no specific focus. All other articles focus on a specific aspect, such as different defect types [6,73]. Teng and Pham [17], for example, propose an SRGM for N-version programming systems. The presented model explicitly considers the error-introduction rate and the error-removal efficiency. Furthermore, this model is able to predict system reliability for N-version programs more accurately than the independent model and can be used to make a decision on when to stop testing. Most often test effort or test resources are considered within the

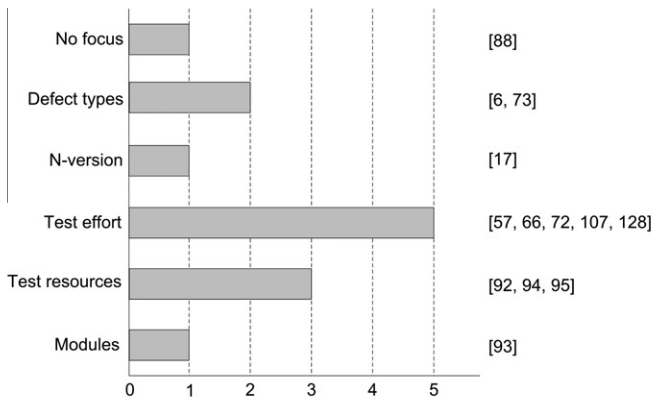

**Fig. 6.** Focus of SRGMs. (See above-mentioned references for further information.)

models. The aim is to improve such models with respect to their precision and the consideration of real context.

Another approach in this category is to predict defect-prone areas. If such areas are identified, testing activities can be focused on these parts. Because not all parts are tested with the same intensity, testing effort can be saved. Twenty-six articles were classified into this category. Most of the articles use metrics for predicting defect-prone parts. Typical metrics are size or complexity. Current releases before and after delivery are considered, as well as historical data. Most often, the predictions are made on the code level; however, some approaches focus on the system level (e.g., [52]).

Fig. 7 gives an overview of the kind of input (i.e., top-level metric) used to perform the predictions; four cases can be distinguished:

1. Product metrics, e.g., size metrics (e.g., lines of code), complexity metrics (e.g., McCabe complexity), or code structure metrics (e.g., number of if-then-else).
2. Process metrics, e.g., development metrics (e.g., number of code changes), or test metrics (e.g., number of test cases).
3. Object-oriented metrics, e.g., weighted method per class, depth of inheritance.
4. Defect metrics, e.g., customer defects, or defects from previous releases.

Some articles use different kinds of top-level metrics. For instance, Sherriff et al. consider product and process metrics [60]. A lot of different detailed metrics are often used when considering one top-level metric (e.g., Tang et al. [27] consider five different object-oriented metrics). Consequently, different approaches and ideas are used to improve the prediction of defect-proneness in order to allocate testing effort in the most suitable manner, and thus, to reduce testing effort.

Of the remaining three articles (see Fig. 5), one suggests using a defect classification that can be used to focus on certain defect classes [83]. Besides mentioning different metrics that can be used to predict defect-prone parts, two articles present approaches on how to evaluate such metrics in order to find the most suitable ones in a given environment [18,27]. Knowing the most appropriate metrics can help to focus on those parts that are most likely to be defect-prone and thus, will save effort compared to an unfocused testing activity.

### *3.2.3. Category test input reduction*

We assigned twenty-two articles to the category test input reduction (see Fig. 8). The techniques and methods discussed in these articles are based on different methods and algorithms.

Twenty of these articles focus on test suite reduction, which is sometimes also called test case reduction or selection. Especially

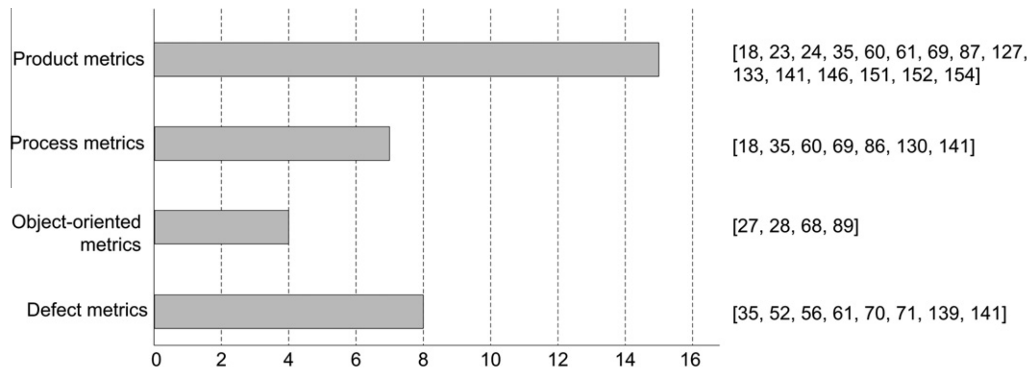

**Fig. 7.** Metrics used in approaches predicting defect-proneness. (See above-mentioned references for further information.)

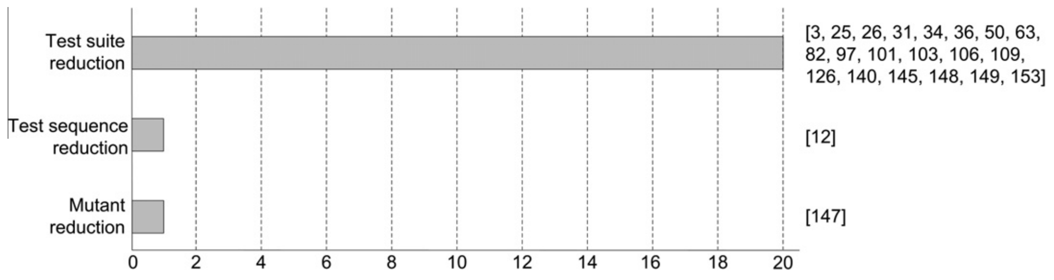

**Fig. 8.** Distribution of articles in the test input reduction category. (See above-mentioned references for further information.)

during regression testing, reducing test cases in a test suite can play an essential role in saving testing costs. A test suite includes not only original test cases but also new ones. Such a test suite contains many redundant test cases. To rerun all the test cases is very expensive and often not efficient.

One article [12] was found that discusses test sequence reduction. This article proposes an approach for eliminating redundancy among test sequences that results in reducing software testing costs. Another article by Usaola and Mateo [147] focuses on reducing the number of mutants during mutation testing.

The twenty articles about test suite reduction mentioned above can be further differentiated.

Two articles present comparisons of different regression test selection techniques [26,63]. In those studies, the relative costs and benefits of five regression test selection techniques are investigated. The studies can be helpful for testers in selecting appropriate regression test selection techniques.

Eighteen articles discuss different test case selection techniques, mainly applied for reducing the set of test cases for regression testing. For example, Pan et al. [50] and Prabhu et al. [109] both present similar techniques for test suite reduction based on modified condition/decision coverage. Both methods consider the coverage degree of test cases for test requirements and the capability of test cases to reveal defects. Smith and Kapfhammer [25] and Ding et al. [34] used greedy algorithms and improved greedy arithmetic as a basis for the proposed test suite reduction techniques. Hla et al. [36] propose an approach for regression test case selection based on prioritization of the test cases. This means that the test cases have to be prioritized to the best new order considering the modified software components. Then the test cases with high priority can be selected for regression tests. Tsai et al. [3] presents a test case selection and ranking technique. The aim of this technique is to select and eliminate test cases with similar coverage, and to rank the test cases according to their potency and coverage. The proposed approach can be applied not just to perform regression testing but also to conduct other tasks, such as N-version programming, web services testing, or standard-based testing.

Two other articles [31,82] also propose test suite reduction techniques, but not directly for regression testing. Woo et al. [31] present a domain-specific approach for test suite reduction with respect to re-targeted compilers. Baudry et al. [82] present a method for test case selection considering fault localization.

Consequently, the articles found use a very heterogeneous set of techniques. Table 3 gives an overview of these approaches.

**Table 3**
Test suite reduction approaches.

| No. | Approach | Reference |
|---|---|---|
| 1 | Model-based approach, statistical calculations considering historical data | [3] |
| 2 | Approach using greedy algorithms | [25] |
| 3 | High-level abstraction approach | [31] |
| 4 | Approach using greedy algorithms | [34] |
| 5 | Particle swarm optimization algorithm | [36] |
| 6 | Bi-object model approach based on MC/DC coverage criterion | [50] |
| 7 | Dynamic basic block approach | [82] |
| 8 | Textual differencing approach | [97] |
| 9 | Model-checker based approach | [101] |
| 10 | Model-based approach using UML Activity diagrams | [103] |
| 11 | Semantic change algorithm and pattern-based approach | [106] |
| 12 | Bi-object model approach based on MC/DC coverage criterion | [109] |
| 13 | Control-flow graph based approach | [126] |
| 14 | Approach using genetic algorithms considering a test history | [140] |
| 15 | Merging approach of particular test case pairs | [145] |
| 16 | Multi-objective approach considering selective coverage of test requirements | [148] |
| 17 | Static path algorithm approach | [149] |
| 18 | Approach using random testing methods, invariants, and genetic algorithms | [153] |

### 3.2.4. Category QA before testing

Seven articles [2,4,29,65,84,116,124] were found that can be classified into the category QA before testing. Those articles discuss different approaches that help to reduce the overall testing effort. Two articles [2,4] were written by Kim et al. The first article [2] presents a case study where the authors investigated how software testing can be improved through code reviews and analysis of code quality. They analyzed how testing activities can be optimized by considering the improvements obtained from code review and analysis. Furthermore, they analyzed the characteristics of software and the current project situation in order to adjust the level of test case design. In the second article [4], Kim et al. present an extended Software Failure Mode Effect Analysis (SW-FMEA) model. Both proposed methods can be used to identify and prevent defects during early development stages. Consequently, defects are found and corrected early, and subsequent costs for testing are saved.

Two articles [116,84] investigate the interactions between software inspections and testing activities, meaning that the possible benefits obtained from bundling software inspections and testing are explored. The article written by Gupta and Jalote [116] presents an experiment where two possible execution orders of applying code inspection and unit testing were compared. The results of this experiment indicate that execution order does not affect the code inspection, but unit tests can be more efficiently and effectively applied after code inspection. Winkler et al. [84] focus on the question of whether software inspections support test case generation. The results of this study show that the collaboration between software inspections and software testing can affect defect detection performance and test case generation positively based on the inspection results. The results of this study can be helpful in supporting project and quality managers in defining inspection and testing effort more precisely.

Laitenberger and Debaud [65] performed a survey in the area of software inspection. They analyzed the entire work done in this area. In particular, they propose a taxonomy that utilizes a generic development life-cycle to contextualize software inspection in detail. This taxonomy can be helpful for practitioners who need to make a decision about which inspection method to choose for a particular development life-cycle stage. Gegick and Williams [124] investigated the correlation between findings generated by static source code analyzers and vulnerabilities discovered by manual analyses and testing. The results show that the alerts generated by source code analyzers can identify high-risk components in the software system early on. Furthermore, the results can be helpful for software engineers who need to make risk management decisions that consider the prioritization of redesign, inspections, and testing efforts. Finally, Wagner et al. [29] compared two static analysis tools and claimed that the output could be used to focus testing effort; however, the defects found by the tool were different from real field defects in a given context.

### 3.2.5. Category test strategy

The category test strategy includes three articles [32,123,150]. Wojcicki and Strooper [32] propose a systematic strategy for selecting certain verification and validation (V&V) combinations. For the user, it is important to know how the available V&V techniques should be combined in order to make testing activities effective and efficient. Perez and Kaiser [123] propose a new test-level integration approach that allows reusing test cases from higher levels at lower test levels. Afzal et al. [150] compare different search-based techniques for predicting the improvement potential in different testing phases. They propose using defect-slippage results for a more suitable allocation of effort to different testing activities.

## 3.3. RQ3: How many existing optimization approaches had been evaluated and how had they been evaluated?

We extracted the relevant data about the evaluation of the selected approaches and techniques to answer research question three. In the first step, we divided the articles into two groups: evaluated and unevaluated. Of the 144 articles, 103 articles (72%) were assigned to the evaluated group, and 41 (28%) to the unevaluated group. For the evaluated group, we further broke down the evaluation by context (industrial or academic) and by method (experience, experiment, case study, or empirical study). Almost 50% of these articles did not provide information about the context in which the evaluation had been performed. Furthermore, in most cases, experiments and case studies had been carried out. The results of this investigation are presented in Fig. 9, which summarizes the main findings.

Compared to the general overview regarding evaluations, Table 4 presents more detailed results with respect to each of the categories. The first two columns describe the main category and, if appropriate, the sub-category. Evaluation category 1 shows how many approaches were evaluated, and, if an approach was evaluated, its context. Evaluation category 2 shows the research method that was applied during evaluations. For the categorization of the research method, we considered the wording from the articles found, which might be designated incorrectly [110]. Furthermore, the sum of the articles counted for all sub-categories may be higher than for the category itself because some arti-

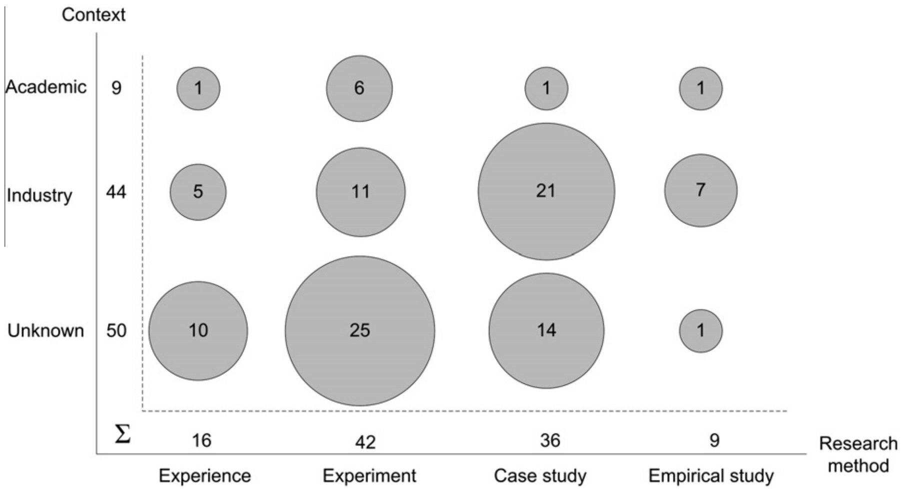

**Fig. 9.** Evaluation of selected approaches.

**Table 4**
Evaluation of effort reduction approaches.

| Category | | Evaluation category I | | | | Evaluation category II | | | |
|---|---|---|---|---|---|---|---|---|---|
| I | II | No | Yes | | | Experience | Experiment | Case study | Empirical study |
| | | | Industrial | Academic | ? | | | | |
| Test automation | Preparation | 24 | 6 | 4 | 26 | 8 | 15 | 10 | 3 |
| Test automation | Execution | 6 | 3 | 3 | 6 | 2 | 4 | 5 | 1 |
| Test automation | Analysis | 3 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Automation | | 28 | 9 | 4 | 30 | 9 | 18 | 13 | 3 |
| Prediction | Defect content | 3 | 4 | 3 | 4 | 3 | 8 | 0 | 0 |
| Prediction | Defect-proneness | 3 | 23 | 0 | 0 | 2 | 4 | 14 | 3 |
| Prediction | Defect classification | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Prediction | | 6 | 27 | 3 | 5 | 6 | 12 | 14 | 3 |
| Test input reduction | Test suite reduction | 2 | 4 | 2 | 12 | 1 | 10 | 5 | 2 |
| Test input reduction | Test sequence reduction | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| Test input reduction | Mutation reduction | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Test input reduction | | 3 | 4 | 2 | 13 | 1 | 10 | 6 | 2 |
| QA before testing | | 3 | 2 | 0 | 2 | 0 | 2 | 2 | 0 |
| Test strategy | | 1 | 2 | 0 | 0 | 0 | 0 | 1 | 1 |

**Table 5**
Distribution of articles by year and class.

| Category | 1991–1995 | 1996–2000 | 2001–2005 | 2006–2010 | Total |
|---|---|---|---|---|---|
| Test automation | 4 | 5 | 16 | 46 | 71 |
| Prediction | 3 | 3 | 13 | 22 | 41 |
| Test input reduction | 1 | 3 | 2 | 16 | 22 |
| QA before testing | 0 | 1 | 0 | 6 | 7 |
| Test strategy | 0 | 0 | 0 | 3 | 3 |
| Total | 8 | 12 | 31 | 93 | 144 |
| %: | 5.6 | 8.3 | 21.5 | 64.6 | |

cles were categorized into more than one sub-category (e.g., see category test automation).

First of all, the category "test automation", which contains the highest number of articles, shows poor evaluation results. Only nine approaches had been evaluated in an industrial setting, whereas 30 articles provide no information. Based on a scoring scale for relevance provided by Ivarsson and Gorschek [137], the contribution could be considered as rather low. The category "prediction" shows the most positive results, as 27 articles provide industrial evaluation results, most of them case studies or experiments. Most of them focus on defect-proneness predictions, i.e., predicting areas where more defects are expected. One main conclusion is that metrics could be used to focus test efforts. However, no universal metric exists that fits best in all contexts. This means that the best metrics have to be identified again in each new context. Moreover, concrete data about specific effort savings are hardly given. The category "test input reduction" is similar to the "test automation" category. The context is often unclear and thus, conclusions could hardly be generalized. Consequently, a lot more sound evaluations are necessary. The remaining two categories, both containing a very low number of articles, present initial results, which were positive, but could not be generalized due to a lack of replications.

### 3.4. RQ4: When were existing optimization approaches published and which publication channels were used?

The purpose of research question 4 was to investigate when existing optimization approaches were published and in which different publication channels. Table 5 presents the distribution of the identified optimization approaches between 1991 and 2010 in intervals of 5 years. It can be observed that until 2000, less attention was given to test optimization approaches and methods and starting from 2001, more articles about reducing testing effort were published. Therefore, it can be concluded that in recent years testing optimization has received increased interest, and that this area has large potential for future research. As we conducted the analysis until 2010, we did not include articles from 2011.

Fig. 10 shows an overview of the concentration of publications with respect to different publication channels. The articles about testing optimization were published in numerous different journals, conferences, symposiums, and workshops. Most of the articles were published across different publication channels. This fact is demonstrated by the biggest circle, which shows that in 86 different publication channels, only one article was published. The next circle indicates that in each of the nine publication channels mentioned (e.g., ESEM, FATES), two articles were published, and so on. The last circles demonstrate the main concentration of publications with respect to test effort reduction approaches. These are the International Conference on Software Engineering (ICSE, nine articles, i.e., nine relevant articles were published at ICSE during the given timeframe) and IEEE Transactions on Software Engineering (TSE, six articles). It can be concluded that this research field is very heterogeneous. Many different aspects are considered, which are published at different conferences, symposiums and workshops, and in different journals.

### 3.5. Threats to validity

Different factors may influence the results of a systematic mapping study, such as the researchers who performed the study, the databases selected, the search string created, and the time constraints selected. The results become more acceptable and

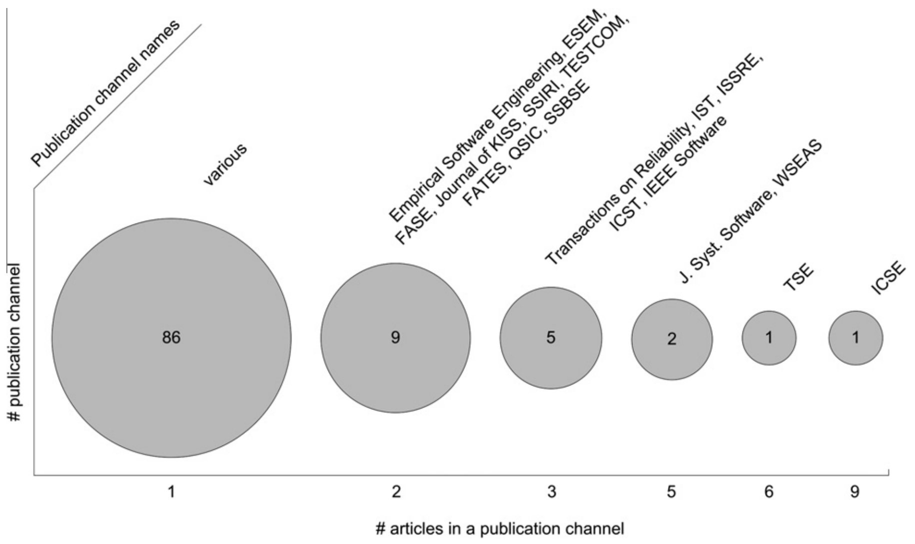

**Fig. 10.** Distribution of articles per number of publication channels.

accurate when these threats to validity are considered, which is done in the next paragraphs.

### 3.5.1. Conclusion validity

We performed a systematic mapping study and every step performed was clearly described. Each step was presented in detail, which allows other researchers to repeat the mapping study.

However, if the selection of articles had been performed by different researchers, it is likely that some articles that were excluded would be included and vice versa, because the decision about the exclusion or inclusion of a certain article depends on the researchers who performed the mapping study. However, it is highly unlikely that these differences based on personal assessments would change the main conclusions drawn from the identified set of articles, which is a general classification of approaches.

### 3.5.2. Construct validity

The aim of the performed mapping study was to obtain a set of relevant articles covering the given research topic. The result set should be as complete as possible. For this reason, we derived the search string systematically. While creating a search string, not all appropriate words are used due to the number of relevant articles found using a search string. For example, the word "verification" was included in the search string and "validation" was excluded. The term "validation" is only a synonym for the term "testing", which was used instead and considered as being more precise in the context of this study. Furthermore, certain terms, such as "validation", are used inconsistently in literature. "Validation" sometimes does not consider only testing activities, but all quality assurance activities along the software development cycle (including inspections or reviews), which is beyond the scope of this mapping study. In conclusion, different or additional terms used in the search string might have resulted in a different set of final articles; however, this would only have a minor influence on the general classification derived, and further articles could easily be categorized according to the presented classification.

### 3.5.3. Internal validity

We presented all inclusion and exclusion criteria. In this mapping study, the selection of relevant articles was performed by two researchers working independently. It is possible that some relevant articles were not included due to the different degree of understanding between the researchers. However, the results of the pilot selection of the relevant articles show that both researchers possessed a high level of agreement (the result of Cohen's Kappa value was about 0.65).

### 3.5.4. External validity

The references within the relevant articles were not examined, even though additional articles might have been found had these references been examined. However, experience has shown that most such articles discuss other aspects of the found techniques.

Furthermore, the classification presented here can only be used in the given context. These results can serve as a starting point for researchers and practitioners working in this field.

## 4. Discussion and implications

### 4.1. General findings

The results of the systematic mapping study confirm that the reduction of test effort during testing is a major goal in modern software development, as the costs for testing are often high. The topic has received increased attention in recent years, which was demonstrated by the number of recent publications.

Automation is an approach that is widely applied for reducing test effort. The automation of certain test process steps, such as creating test cases or executing them, can reduce the amount of testing time. Furthermore, prediction approaches and test input reduction approaches are widely used to save test effort. Prediction approaches support the focusing of testing activities and avoid or reduce the need for each part of a system to be tested with the same effort. As defects are often not distributed equally, the test effort can be reduced by such an approach. Test input reduction approaches mainly support reducing the number of test cases when regression testing is conducted, whereas the same number of defects should be found with the reduced set of test cases. Quality assurance activities conducted before testing, which reduce the number of defects proactively, are the fourth approach identified to reduce test effort. However, only a small set of articles was identified that support the combination of early quality assurance and testing.

### 4.2. Implications for practitioners

We presented existing approaches, methods, and techniques that aim at reducing effort when conducting test activities. In addition, we illustrated existing evidence about the approaches that show their maturity. Our classification indicates whether an approach is an established solution applied in an industrial context or merely in academic environments. Automation approaches are one major solution for reducing test effort, although concrete effort reductions are often unclear. Consequently, practitioners have to decide carefully what and how to automate. Furthermore, concentrating testing on parts that are expected to be defect-prone is another big trend. The current evaluation results imply that at least one metric can be identified in each context that will lead to appropriate prediction results. However, such an analysis may take some time and requires the collection of metrics. New trends, such as the use of defect data from static quality assurance activities that are available early, may refine such approaches. In general, practitioners can get an idea of which approaches and techniques may be suitable for optimizing testing activities in their own context. However, in this case, adaptations of the techniques may be necessary and the effort improvement has to be analyzed in order to assess the benefit in the given context.

### 4.3. Implications for researchers

The main objective of this systematic mapping study was to provide an overview of existing improvement approaches that reduce the effort required for testing activities. Only a small number of approaches could be found where quality assurance activities performed before testing were conducted in order to test effort. It may be worthwhile to concentrate on this direction for future research in order to further improve this kind of efficiency, as it is well known that early defect detection is able to reduce costs. Furthermore, adaptation of existing approaches to new contexts and their specific requirements is another implication for future research. In addition, more evidence is necessary for assessing the approaches. This includes, for example, replications of studies and a more concise definition of the context in order to make results comparable. Finally, combinations of improvement approaches could lead to much greater improvements in effort reduction.

## 5. Conclusion

We performed a mapping study and presented the state of the art regarding existing approaches that are able to reduce the effort of testing techniques.

During the mapping study, 144 articles were found that describe different approaches and techniques for reducing testing effort. We classified the articles into five different categories: test automation, prediction, test input reduction, quality assurance before testing, and test strategy. The numbers of articles per category are highly uneven. Fifty percent of the articles discuss test automation, and another 30% describe prediction approaches. Predictions determine, for example, when to stop testing or which modules or classes are defect-prone. They can support a test manager in deciding, e.g., how much testing effort is required. The category test input reduction contains about 15% of all articles and includes mainly articles that present different test case selection and prioritization techniques, as well as methods for optimizing and reducing test suites. Quality assurance before testing is only covered by a few of the relevant articles. In these articles, results from static quality assurance activities, such as code inspections and reviews, are used to reduce effort for later testing activities. Three other articles were classified as test strategy.

While performing the mapping study, the starting year for including articles was set to 1991 to ensure that the most relevant research works would be included, and the last year for inclusion were publications from 2010. Until 2000, less attention was given to test optimization approaches and methods, and starting from 2001, more articles about reducing testing effort were published. Furthermore, it can be concluded that in recent years, testing optimization has received increased interest. In relative numbers, test input reduction and quality assurance before testing had the highest increase during the past 5 years, followed by test automation and test strategy. Consequently, this research is of high interest and various results are expected in the future. This is also reflected by the high number of different publication channels.

About 70% of the approaches found in the articles had been evaluated. However, the information about the context in which the evaluation had been performed (i.e., industrial or academic) was not clearly stated in all categories except prediction. Though a large number of evaluations could be identified, their rigor and relevance seem to be rather poor based on our initial analysis, and many more sound evaluations are necessary to generalize the conclusions drawn. Furthermore, very little explicit effort reduction data were found. Thus, conclusions about the potential of these approaches can often not be determined and generalized.

In summary, it can be stated that there are numerous approaches to reducing test effort. Only few articles were found that present a combination of static and dynamic quality assurance activities, for example a combination of software inspections and testing. The synergies that might emerge from the combination of these two techniques remain unexplored.

Based on the results of this mapping study, the objective of one of our current research projects is to develop an integrated approach that describes how inspection and testing techniques can be applied together to reduce the overall costs for testing. Specifically, we want to investigate the relationships between inspections and testing. One goal is to draw conclusions on how inspection results can improve the prediction of defect-prone parts for focusing testing activities [96,131]. Such an integrated approach should be able to reduce testing effort.

## Acknowledgments

Parts of this work have been funded by the Stiftung Rheinland-Pfalz für Innovation project "Qualitäts-KIT" (Grant No. 925). We would like to thank Sonnhild Namingha for proofreading.

## References

[1] A. Bertolino, R. Mirandola, E. Peciola, A case study in branch testing automation, in: Third International Conference on Achieving Quality in Software, 1997, pp. 47–59.

[2] K. Young, H. Soo, A case study on the improvement of software test effectiveness through static testing, Journal of KISS: Software and Applications 34 (March) (2007) 212–218.

[3] W. Tsai, Xinyu Zhou, R. Paul, Yinong Chen, Xiaoying Bai, A coverage relationship model for test case selection and ranking for multi-version software, in: IEEE International Symposium on High Assurance, Systems Engineering, 2007, pp. 105–112.

[4] H. Kim, H. Han, A defect prevention model based on SW-FMEA, Journal of KISS: Software and Applications 33 (July) (2006) 605–614.

[5] T. Xie and J. Zhao, A framework and tool supports for generating test inputs of AspectJ programs, in: 5th International Conference on Aspect-oriented Software, Development, 2006, pp. 190–201.

[6] O. Shatnawi, P. Kapur, A generalized software fault classification model, WSEAS Transactions on Computers 7 (2008) 1375–1384.

[7] F. Belli, A. Hollmann, M. Kleinselbeck, A graph-model-based testing method compared with the classification tree method for test case generation, in: 3rd IEEE International Conference on Secure Software Integration Reliability Improvement, 2009, pp. 193–200.

[8] M. Penaloza, A. Logar, J. Johnson, M. Boucher, A Java unit and integration testing tool, in: Proceedings of the ISCA 16th International Conference Computers and their Applications, 2001, pp. 358–61.
[9] A. Boklund, C. Selvefors, A low budget approach to distributed automated black-box testing, in: International Conference on Software Engineering Research and, Practice, 2005, pp. 302–308.
[10] P. Neto, R. Resende, C. Padua, A method for information systems testing automation, in: 17th International Conference on Advanced Information Systems Engineering, 2005, pp. 504–18.
[11] E. Diaz, J. Tuya, R. Blanco, A modular tool for automated coverage in software testing, in: Proceedings. Eleventh Annual International Workshop on Software Technology and Engineering, Practice, 2004, pp. 241–246.
[12] H. Miao, P. Liu, J. Mei, H. Zeng, A new approach to automated redundancy reduction for test sequences, in: 15th IEEE Pacific Rim International Symposium on Dependable, Computing, 2009, pp. 93–98.
[13] Y. Cui, L. Li, and S. Yao, A new strategy for pairwise test case generation, in: 3rd International Symposium on Intelligent Information Technology Application, 2009, pp. 303–306.
[14] B. Biswal, P. Nanda, D. Mohapatra, A novel approach for scenario-based test case generation, International Conference on Information Technology 2008 (2008) 244–247.
[15] M. Catelani, L. Ciani, V. Scarano, A. Bacioccola, A novel approach to automated testing to increase software reliability, in: IEEE Instrumentation and Measurement Technology Conference, 2008, pp. 1499–502.
[16] L. Briones, E. Brinksma, M. Stoelinga, A semantic framework for test coverage, in: 4th International, Symposium on Automated Technology for Verification and Analysis, 2006, pp. 399–414.
[17] X. Teng, H. Pham, A software-reliability growth model for N-version programming systems, IEEE Transactions on Reliability 51 (2002) 311–321.
[18] E. Arisholm, L. Briand, E. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, Journal of Systems and Software 83 (January) (2010) 2–17.
[19] S. Ali, L. Briand, H. Hemmati, R. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test case generation, IEEE Transactions on Software Engineering 99 (2010) 1.
[20] E. Martins, M. de Fatima Mattiello-Francisco, A tool for fault injection and conformance testing of distributed systems, in: First Latin-American, Symposium on Dependable Computing, 2003, pp. 282–302.
[21] M. Gallagher, V. Lakshmi Narasimhan, ADTEST: a test data generation suite for Ada software systems, IEEE Transactions on Software Engineering 23 (1997) 473–484.
[22] F. Wang, S. Wang, Y. Ji, An automatic generation method of executable test case using model-driven architecture, in: 4th International Conference on Innovative Computing, Information and Control, 2009, pp. 389–393.
[23] A.G. Koru, J. Tian, An empirical comparison and characterization of high defect and high complexity modules, Journal of Systems and Software 67 (2003) 153–163.
[24] Y. Shin, L. Williams, An empirical model to predict security vulnerabilities using code complexity metrics, in: 2nd International Symposium on Empirical Software Engineering and, Measurement, 2008, pp. 315–317.
[25] A.M. Smith, G.M. Kapfhammer, An empirical study of incorporating cost into test suite reduction and prioritization, in: 24th Annual ACM Symposium on Applied, Computing, 2009, pp. 461–467.
[26] T. Graves, M. Harrold, J. Kim, A. Porter, G. Rothermel, An empirical study of regression test selection techniques, ACM Transactions on Software Engineering and Methodology 10 (2001) 184–208.
[27] M. Tang, M. Kao, M. Chen, An empirical study on object-oriented metrics, in: Proceedings of METRICS '99: Sixth International Symposium on Software Metrics, 1999, pp. 242–249.
[28] H.M. Olague, L.H. Etzkorn, S.L. Messimer, H.S. Delugach, An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study, Journal of Software Maintenance and Evolution 20 (2008) 171–197.
[29] S. Wagner, F. Deissenboeck, M. Aichner, J. Wimmer, M. Schwalb, An evaluation of two bug pattern tools for Java, in: 1st International Conference on Software Testing, Verification and Validation, 2008, pp. 248–257.
[30] J. Sant, A. Souter, L. Greenwald, An exploration of statistical models for automated test case generation, in: Proceedings of the Third International Workshop on Dynamic, Analysis, 2005, pp. 1–7.
[31] G. Woo, H. Seok Chae, H. Jang, An intermediate representation approach to reducing test suites for retargeted compilers, in: 12th Ada-Europe International Conference on Reliable Software Technologies, 2007, pp. 100–113.
[32] M.A. Wojcicki, P. Strooper, An iterative empirical strategy for the systematic selection of a combination of verification and validation technologies, in: 5th International Workshop on Software, Quality, 2007, p. 9.
[33] L. Kawakami, A. Knabben, D. Rechia, D. Bastos, O. Pereira, R. Pereira e Silva, L.C. Dos Santos, An object-oriented framework for improving software reuse on automated testing of mobile phones, in: 19th IFIP TC6/WG6.1 International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing Software, 2007, pp. 199–211.
[34] W. Ding, J. Kou, K. Li, Z. Yang, An optimization method of test suite in regression test model, in: WRI World Congress on Software Engineering, 2009, pp. 180–183.
[35] R. Selby, V. Basili, Analyzing error-prone system structure, IEEE Transactions on Software Engineering 17 (1991) 141–152.
[36] K.H.S. Hla, Y.S. Choi, J.S. Park, Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting, in: 8th IEEE International Conference on Computer and Information Technology Workshops, 2008, pp. 527–532.
[37] N. Oster, Automated generation and evaluation of dataflow-based test data for object-oriented software, in: 1st International Conference on the Quality of Software Architectures and 2nd International Workshop on Software, Quality, 2005, pp. 212–226.
[38] D. Mohapatra, P. Bhuyan, D.P. Mohapatra, Automated test case generation and its optimization for path testing using genetic algorithm and sampling, in: WASE International Conference on Information, Engineering, 2009, pp. 643–646.
[39] M. Harman, F. Islam, T. Xie, S. Wappler, Automated test data generation for aspect-oriented programs, in: 8th ACM International Conference on Aspect-Oriented Software, Development, 2009, pp. 185–196.
[40] R. Gupta, M. Soffa, Automatic generation of a compact test suite, Algorithms, Software, Architecture. Information Processing, in: IFIP 12th World Computer Congress, 1992, pp. 237–243.
[41] H.K. Seung, S.K. Hyeon, Automatic generation of testing environments for web applications, in: International Conference on Computer Science and, Software Engineering, 2008, pp. 694–697.
[42] R. Romli, S. Sulaiman, K. Zamli, Automatic programming assessment and test data generation a review on its approaches, in: International Symposium in Information Technology, 2010, pp. 1186–1192.
[43] A. Khamis, M. Girgis, A. Ghiduk, Automatic software test data generation for spanning sets coverage using genetic algorithms, Computing and Informatics 26 (2007) 383–401.
[44] B. Baudry, F. Fleurey, J. Jezequel, Y. Le Traon, Automatic test case optimization: a bacteriologic algorithm, IEEE Software 22 (2005) 76–82.
[45] N. Oster, F. Saglietti, Automatic test data generation by multi-objective optimisation, in: 25th International Conference on Computer Safety Reliability, and Security, 2006, pp. 426–38.
[46] P. Bokil, P. Darke, U. Shrotri, R. Venkatesh, Automatic test data generation for C programs, in: Third IEEE International Conference on Secure Software Integration and Reliability Improvement, 2009, pp. 359–368.
[47] D.L. Barbosa, H.S. Lima, P.D.L. Machado, J.C.A. Figueiredo, M.A. Juca, W.L. Andrade, Automating functional testing of components from UML specifications, International Journal of Software Engineering and Knowledge Engineering 17 (2007) 339–358.
[48] K. Im, T. Im, J.D. McGregor, Automating test case definition using a domain specific language, in: 46th Annual Southeast Regional Conference on XX, 2008, pp. 180–185.
[49] M. Vieira, J. Leduc, B. Hasling, R. Subramanyan, and J. Kazmeier, Automation of GUI testing using a model-driven approach, in Proceedings of the 2006 international workshop on Automation of software test, 2006, p. 9.
[50] L. Pan, B. Zou, J. Li, H. Chen, Bi-objective model for test-suite reduction based on modified condition/decision coverage, in: Proceedings. 11th Pacific Rim International Symposium on Dependable, Computing, 2005, p. 7.
[51] R. Zhao, Z. Li, Boundary value testing using integrated circuit fault detection rule, in: Testing: Academic and Industrial Conference – Practice and Research, Techniques, 2009, pp. 3–12.
[52] E. Moritz, Case study: how analysis of customer found defects can be used by system test to improve quality, in: 31st International Conference on, Software Engineering, 2009, pp. 123–129.
[53] J.J. Li, D. Weiss, H. Yee, Code-coverage guided prioritized test generation, Information and Software Technology 48 (2006) 1187–1198.
[54] R.S. Sisodia, V. Channakeshava, Combinatorial approach for automated platform diversity testing, in: 4th International Conference on Software, Engineering Advances, 2009, pp. 134–139.
[55] R. Kuhn, R. Kacker, Yu Lei, J. Hunter, Combinatorial software testing, Computer 42 (2009) 94–96.
[56] M. Hamill, K. Goseva-Popstojanova, Common trends in software fault and failure data, IEEE Transactions on Software Engineering 35 (2009) 484–496.
[57] C. Huang, Cost-reliability-optimal release policy for software reliability models incorporating improvements in testing efficiency, Journal of Systems and Software 77 (2005) 139–155.
[58] Z. Li, S. Lu, S. Myagmar, Y. Zhou, CP-Miner: a tool for finding copy-paste and related bugs in operating system code, in: Proceedings of the Sixth Symposium on Operating Systems Design and Implementation, 2004, pp. 289–302.
[59] G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, Z. Su, Dynamic test input generation for web applications, in: Proceedings of the 2008 International Symposium on Software Testing and, Analysis, 2008, p. 249.
[60] M. Sherriff, N. Nagappan, L. Williams, M. Vouk, Early estimation of defect density using an in-process Haskell metrics model, in: 1st International Workshop on Advances in Model-Based Testing, 2005.
[61] N. Ohlsson, A.C. Eriksson, M. Helander, Early risk-management by identification of fault-prone modules, Empirical Software Engineering 2 (1997) 166–173.
[62] W.E. Wong, Y. Lei, X. Ma, Effective generation of test sequences for structural testing of concurrent programs, in: 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005, pp. 539–548.
[63] T.L. Graves, M.J. Harrold, J. Kim, A. Porter, G. Rothermel, Empirical study of regression test selection techniques, Proceedings of the, International Conference on Software Engineering 1998 (1998) 188–197.

[64] R. Gupta, M. Soffa, Employing static information in the generation of test cases, Software Testing, Verification and Reliability 3 (1993) 29–48.
[65] O. Laitenberger, J. Debaud, Encompassing life cycle centric survey of software inspection, Journal of Systems and Software 50 (2000) 5–31.
[66] C. Lin, C. Huang, Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models, Journal of Systems and Software 81 (2008) 1025–1038.
[67] T. Ummu Salima, A. Askarunisha, N. Ramaraj, Enhancing the efficiency of regression testing through intelligent agents, in: International Conference on Computational Intelligence and Multimedia Applications, 2008, pp. 103–108.
[68] S. Kpodjedo, F. Ricca, G. Antoniol, P. Galinier, Evolution and search based metrics to improve defects prediction, in: 1st International Symposium on Search Based, Software Engineering, 2009, pp. 23–32.
[69] P.L. Li, J. Herbsleb, M. Shaw, B. Robinson, Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc., in: 28th International Conference on, Software Engineering, 2006, pp. 413–422.
[70] T. Illes-Seifert, B. Paech, Exploring the relationship of a file's history and its fault-proneness: an empirical method and its application to open source programs, Information and Software Technology 52 (2010) 539–558.
[71] T. Illes-Seifert, B. Paech, Exploring the relationship of history characteristics and defect count: an empirical study, in: Workshop on Defects in Large Software Systems, 2008, pp. 11–15.
[72] S. Kuo, C. Huang, M. Lyu, Framework for modeling software reliability, using various testing-efforts and fault-detection rates, IEEE Transactions on Reliability 50 (2001) 310–320.
[73] O. Shatnawi, Generalized software fault classification model, in: 12th WSEAS International Conference on Computers, 2008, pp. 993–998.
[74] Y. Yu, S. Ng, E. Chan, Generating, selecting and prioritizing test cases from specifications with tool support, in: Proceedings. Third International Conference on Quality Software, 2003, pp. 83–90.
[75] B. Kitchenham, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version 2.3. Technical Report S.o.C.S.a.M. Software Engineering Group, Keele University and Department of Computer Science University of Durham, 2007.
[76] G. Fraser, B.K. Aichernig, F. Wotawa, Handling model changes: regression testing and test-suite update with model-checkers, Electronic Notes in Theoretical Computer Science 190 (2007) 33–46.
[77] M.N. Ngo, H.B.K. Tan, Heuristics-based infeasible path detection for dynamic test data generation, Information and Software Technology 50 (2008) 641–655.
[78] L. Huang, B. Boehm, How much software quality investment is enough: a value-based approach, IEEE Software 23 (2006) 88–95.
[79] A. Beer, S. Mohacsi, C. Stary, IDATG: an open tool for automated testing of interactive software, in: Proceedings of the 1998 IEEE 22nd Annual International Computer Software & Applications Conference, 1998, pp. 470–475.
[80] IEEE Xplore, <http://ieeexplore.ieee.org/search/advsearch.jsp>.
[81] R. Gerlich, R. Gerlich, T. Boll, P. Chevalley, Improving test automation by deterministic methods in statistical testing, in: DASIA: Data Systems In Aerospace, 2006, p. Eurospace.
[82] B. Baudry, F. Fleurey, Y. Le Traon, Improving test suites for efficient fault localization, in: 28th International Conference on, Software Engineering, 2006, pp. 82–91.
[83] J. Chaar, M. Halliday, I. Bhandari, R. Chillarege, In-process evaluation for software inspection and test, IEEE Transactions on Software Engineering 19 (11) (1993) 1055–1070.
[84] D. Winkler, S. Biffl, K. Faderl, Investigating the temporal behavior of defect detection in software inspection and inspection-based testing, in: 11th International Conference on Product-Focused Software Process Improvement, 2010, pp. 17–31.
[85] A. Calvagna, A. Gargantini, IPO-s: incremental generation of combinatorial interaction test data based on symmetries of covering arrays, in: IEEE International Conference on Software Testing, Verification, and Validation Workshops, 2009, pp. 10–18.
[86] R. Paul, Metrics based classification trees for software test monitoring and management, in: Proceedings Sixth International Conference on Tools with, Artificial Intelligence, 1994, pp. 534–540.
[87] J. Michura, M. Capretz, Metrics suite for class complexity, in: Proceedings. ITCC International Conference on Information Technology: Coding and, Computing, 2005, pp. 404–409.
[88] C.Y. Htoon, N.L. Thein, Model-based testing considering cost, reliability and software quality, in: 6th Asia-Pacific Symposium on Information and Telecommunication Technologies, 2005, pp. 160–164.
[89] I. Bluemke, Object oriented metrics useful in the prediction of class testing complexity, in: Proceedings of the 27th EUROMICRO Conference, 2001, pp. 130–136.
[90] P.R.F. Nunes, S. Hanazumi, A.C.V. De Melo, OConGraX – Automatically generating data-flow test cases for fault-tolerant systems, in: 21st IFIP International Conference on Testing of Communicating Systems and 9th International Workshop on Formal Approaches to Testing of Software, 2009, pp. 229–234.
[91] R. Lea, S. Chen, and C. Chung, On generating test data from prototypes, in: Proceedings of the Fifteenth Annual International Computer Software and Applications Conference, 1991, pp. 345–350.
[92] C. Huang, J. Lo, S. Kuo, M. Lyu, Optimal allocation of testing resources for modular software systems, in: Proceedings 13th International Symposium on Software, Reliability Engineering, 2002, pp. 129–138.
[93] C. Huang, J. Lo, J. Lin, C. Sue, C. Lin, Optimal resource allocation and sensitivity analysis for modular software testing, in: Proceedings. IEEE Fifth International Symposium on Multimedia, Software Engineering, 2003, pp. 231–238.
[94] P. Jha, D. Gupta, Bo Yang, P. Kapur, Optimal testing resource allocation during module testing considering cost, testing effort and reliability, Computers & Industrial Engineering 57 (2009) 1122–1130.
[95] C. Huang, M.R. Lyu, Optimal testing resource allocation, and sensitivity analysis in software development, IEEE Transactions on Reliability 54 (4) (2005) 592–603.
[96] F. Elberzhager, J. Muench, D. Rombach, B. Freimut, Optimizing cost and quality by integrating inspection and test processes, in: Proceedings of the International Conference on Software and Systems Process, 2011, pp. 3–12.
[97] F. Vokolos, P. Frankl, Pythia: a regression test selection tool based on textual differencing, in: Proceedings of 3rd International Conference on Reliability, Quality and Safety of Software-Intensive, System, 1997, pp. 3–21.
[98] M. Marre, A. Bertolino, Reducing and estimating the cost of test coverage criteria, in: Proceedings of IEEE 18th International Conference on, Software Engineering, 1996, pp. 486–94.
[99] V. Jagannath, Yun Young Lee, B. Daniel, D. Marinov, Reducing the costs of bounded-exhaustive testing, in: 12th International Conference on Fundamental Approaches to Software Engineering, 2009, pp. 171–185.
[100] D. Brenner, C. Atkinson, R. Malaka, M. Merdes, B. Paech, D. Suliman, Reducing verification effort in component-based software engineering through built-in testing, Information Systems Frontiers 9 (2) (2007) 151–162.
[101] G. Fraser, F. Wotawa, Redundancy based test-suite reduction, in: 10th International Conference on Fundamental Approaches to, Software Engineering, 2007, pp. 291–305.
[102] N. Li, T. Xie, N. Tillmann, J. de Halleux, W. Schulte, Reggae: Automated test generation for programs using complex regular expressions, in: 2009 24th IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 515–519.
[103] E. Martins, V. G. Vieira, Regression test selection for testable classes, in: 5th European Dependable Computing Conference, 2005, pp. 453–470.
[104] Q. Xie, M. Grechanik, C. Fu, REST: a tool for reducing effort in script-based testing, in: 24th IEEE International Conference on Software, Maintenance, 2008, pp. 468–469.
[105] W. Tsai, L. Yu, X. Liu, A. Saimi, Y. Xiao, Scenario-based test case generation for state-based embedded systems, in: Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference, 2003, pp. 335–342.
[106] D. Binkley, Semantics guided regression test cost reduction, IEEE Transactions on Software Engineering 23 (8) (1997) 498–516.
[107] C. Huang, J. Lo, S. Kuo, M.R. Lyu, Software reliability modeling and cost estimation incorporating testing-effort and efficiency, in: Proceedings of the 1999 10th International Symposium on Software Reliability Engineering, 1999, pp. 62–72.
[108] M.J. Gallagher, V. Narasimhan, Software test data generation using program instrumentation, in: Proceedings of the IEEE 1st International Conference on Algorithms and Architectures for Parallel Processing, Part 1, 1995, pp. 575–584.
[109] J. Prabhu, N. Malmurugan, G. Gunasekaran, R. Gowtham, Study of ERP test-suite reduction: based on modified condition/decision coverage, in: Second International Conference on Computer Research and, Development, 2010, pp. 373–378.
[110] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in, Software Engineering, 2008, pp. 1–10.
[111] F. Pinte, N. Oster, F. Saglietti, Techniques and tools for the automatic generation of optimal test data at code, model and interface level, in: 30th International Conference on, Software Engineering, 2008, pp. 927–928.
[112] L. Lazic, N. Mastorakis, Techniques to reduce a set of test cases, WSEAS Transactions on Computers 5 (11) (2006) 2813–2826.
[113] P. Saraph, M. Last, A. Kandel, Test case generation and reduction by automated input-output analysis, in: Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics, 2003, pp. 768–773.
[114] Y. Yongfeng, L. Bin, L. Minyan, L. Zhen, Test cases generation for embedded real-time software based on extended UML, in: International Conference on Information Technology and Computer Science, 2009, pp. 69–74.
[115] A. Spillner, Test criteria and coverage measures for software integration testing, Software Quality Journal 4 (4) (1995) 275–286.
[116] A. Gupta, P. Jalote, Test inspected unit or inspect unit tested code? in: First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007, 2007, pp. 51–60.
[117] D. Travison, G. Staneff, Test instrumentation and pattern matching for automatic failure identification, in: 1st International Conference on Software Testing, Verification and Validation, 2008, pp. 377–386.

[118] M. Hirayama, O. Mizuno, T. Kikuno, Test item prioritizing metrics for selective software testing, IEICE Transactions on Information and Systems 87 (12) (2004) 2733–2743.
[119] R. Hewett, P. Kijsanayothin, D. Smavatkul, Test order generation for efficient object-oriented class integration testing, in: 20th International Conference on Software Engineering & Knowledge, Engineering, 2008, pp. 703–708.
[120] C. Baek, S. Park, K. Choi, TEST: an effective automation tool for testing embedded software, WSEAS Transactions on Information Science and Applications 2 (2005) 1214–1219.
[121] Health, Social, and Economic Research, The Economic Impacts of Inadequate Infrastructure for Software Testing, National Institute of Standards and Technology, 2002.
[122] D. Talby, The perceived value of authoring and automating acceptance tests using a model driven development toolset, in: Workshop on Automation of Software, Test, 2009, pp. 154–157.
[123] A. Perez, S. Kaiser, Top-down reuse for multi-level testing, in: 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, 2010, pp. 150–159.
[124] M. Gegick, L. Williams, Toward the use of automated static analysis alerts for early identification of vulnerability- and attack-prone components, in: 2nd International Conference on Internet Monitoring and Protection, 2007.
[125] Z. Li, J. Zhu, L. Zhang, N. Mitsumori, Towards a practical and effective method for Web services test case generation, in: Workshop on Automation of Software, Test, 2009, pp. 106–114.
[126] J. Zhao, T. Xie, N. Li, Towards regression test selection for AspectJ programs, in: 2nd Workshop on Testing Aspect-oriented Programs, 2006, pp. 21–26.
[127] T. Khoshgoftaar, Xiaojing Yuan, E. Allen, W. Jones, J. Hudepohl, Uncertain classification of fault-prone software modules, Empirical Software Engineering 7 (4) (2002) 297–318.
[128] P. Kapur, O. Shatnawi, A. Aggarwal, R. Kumar, Unified framework for developing testing effort dependent software reliability growth models, WSEAS Transactions on Systems 8 (4) (2009) 521–531.
[129] K. Whitmill, Usage based test case generation, in: Proceedings Software Testing Analysis and, Review, 1996, pp. 123–152.
[130] N. Nagappan, L. Williams, M. Vouk, J. Osborne, Using in-process testing metrics to estimate post-release field quality, in: 18th IEEE International Symposium on Software, Reliability Engineering, 2007, pp. 209–214.
[131] F. Elberzhager, R. Eschbach, J. Muench, Using inspection results for prioritizing test activities, in: Proceedings of the 21st International Symposium on Software Reliability Engineering, Supplemental Proceedings, 2010, pp. 263–272.
[132] P. Pocatilu, Using open source software testing tools for automated unit testing, Open Source Science Journal 1 (1) (2009) 163–172.
[133] T. Ostrand, E. Weyuker, R. Bell, Using static analysis to determine where to focus dynamic testing effort, in: Second International Workshop on Dynamic Analysis and 26th International Conference on, Software Engineering, 2004, pp. 1–8.
[134] M. Nita, D. Notkin, White-box approaches for improved testing and analysis of configurable software systems, in: 31st International Conference on, Software Engineering, 2009, pp. 307–310.
[135] Zotero, <http://www.zotero.org/>.
[136] D. Jackson, M. Thomas, L.I. Millett (Eds.). Software for Dependable Systems: Sufficient Evidence? Committee on Certifiably Dependable Software Systems, National Research Council, National Academy of Sciences, 2007.
[137] M. Ivarsson, T. Gorschek, A method for evaluating rigor and industrial relevance of technology evaluations, Empirical Software Engineering 16 (3) (2011) 365–395.
[138] IEEE Standard 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology, 1990.
[139] A.K. Amanpreet, A.S. Brar, P.S. Sandhu, An empirical approach for software fault prediction, in: Fifth International Conference on Industrial and, Information Systems, 2010, pp. 261–265.
[140] S. Nachiyappan, A.Vimaladevi, C.B. SelvaLakshmi, An evolutionary algorithm for regression test suite reduction, in: International Conference on Communication and Computational Intelligence.
[141] M. D'Ambros, M. Lanza, R. Robbes, An extensive comparison of bug prediction approaches, in: 7th IEEE Working Conference on Mining Software Repositories, 2010, pp. 31–41.
[142] D. Chen, X. Li, S. Zhao, Auto-generation and redundancy reduction of test cases for reactive systems, in: 2nd International Conference on Software Technology and, Engineering, 2010, pp. 125–130.
[143] W. Jin, A. Orso, T. Xie, Automated behavioral regression testing, in: Third International Conference on Software Testing, Verification and Validation, 2010, pp. 137–146.
[144] W. Xu, D. Huang, Automated testing for database system, in: International Conference on Biomedical Engineering and Computer Science, 2010, pp. 1–4.
[145] H. Cichos, T.S. Heinze, Efficient reduction of model-based generated test suites through test case pair prioritization, in: Workshop on Model-Driven Engineering, Verification, and Validation, 2010, pp. 37–42.
[146] L. Yunfeng, B. Kerong, Metrics selection for fault-proneness prediction of software modules, in: International Conference on Computer Design And Appliations, 2010, pp. 191–195.
[147] M.P. Usaola, P.R. Mateo, Mutation testing cost reduction techniques: a survey, IEEE Software 27 (3) (2010) 80–86.
[148] Q. Gu1, B. Tang, D.X. Chen, Optimal regression testing based on selective coverage of test requirements, in: International Symposium on Parallel and Distributed Processing with Applications, 2010, pp. 419–426.
[149] B. Jiang, Y. Mu, Z. Zhang, Research of optimization algorithm for path-based regression testing suit, in: Second International Workshop on Education Technology and Computer Science, 2010, pp. 303–306.
[150] W. Afzal, R. Torkar R. Feldt, G. Wikstrand, Search-based prediction of fault-slip-through in large software projects, in: 2nd International Symposium on Search Based, Software Engineering, pp. 79–88.
[151] L.T. Giang, D. Kang, D.H. Bae, Software fault prediction models for web applications, in: 34th Annual IEEE Computer Software and Applications Conference Workshops, pp. 51–56.
[152] R. Lincke, T. Gutzmann, W. Löwe, Software quality prediction models compared, in: 10th International Conference on Quality Software, 2010, pp. 82–91.
[153] N. Pan, F. Zeng, Y.H. Huang, Test case reduction based on program invariant and genetic algorithm, in: 6th International Conference on Wireless Communications Networking and Mobile, Computing, 2010, pp. 1–5.
[154] A.K. Pandey, N.K. Goyal, Test effort optimization by prediction and ranking of fault-prone software modules, in: 2nd International Conference on Reliability, Safety & Hazard, 2010, pp. 136–142.
[155] F.Q.B. da Silva, A.L.M. Santos, S. Soares, A.C.C. Franca, C.V.F. Monteiro, F.F. Maciel, Six years of systematic literature reviews in software engineering: an updated tertiary study, Information and Software Technology 53 (2011) 899–913.
[156] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkmann, Systematic literature reviews in software engineering – a systematic literature review, Information and Software Technology 51 (2009) 7–15.
[157] N. Juristo, A.M. Moreno, S. Vegas, Reviewing 25 years of testing technique experiments, Empirical Software Engineering Journal 1–2 (2004) 7–44.
[158] P. Runeson, C. Andersson, T. Thelin, A. Andrews, T. Berling, What do we know about defect detection methods?, IEEE Software 23 (3) (2006) 82–86
[159] Z. Zakaria, R. Atan, A.A.A. Ghani, N.F.M. Sani, Unit testing approaches for BPEL: a systematic review, in: 16th Asia Pacific, Software Engineering Conference, 2009, pp. 316–322.
[160] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert Systems with Applications 36 (4) (2009) 7346–7354.
[161] B. Haugset, G.K. Hanssen, Automated acceptance testing: a literature review and an industrial case study, Agile, 2008, pp. 27–38.