UNIVERSITY OF GOTHENBURG

Exploring Code Coverage in Software Testing and its Correlation with Software Quality

A Systematic Literature Review

Bachelor of Science Thesis in Software Engineering and Management

JOY W. HOLLÉN

PATRICK S. ZACARIAS

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, August 2013

Exploring Code Coverage in Software Testing and its Correlation with Software Quality

A Systematic Literature Review

JOY W. HOLLÉN

PATRICK S. ZACARIAS

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, August 2013

Exploring Code Coverage in Software Testing and its Correlation with Software Quality; A Systematic Literature Review

Joy W. Hollén
Software Engineering and Management
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden
gustrawi@student.gu.se

Patrick S. Zacarias
Software Engineering and Management
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden
gussubpa@student.gu.se

Abstract

As the world around us is increasingly becoming defined by software and the size and complexity of the software systems increases, as well as our dependance on them, it becomes all the more important that the software is thoroughly tested in an attempt to guarantee high reliability. In order to measure the thoroughness of testing, code coverage is often applied as an attempt at measuring the software testing. The paper aims to explore how code coverage is correlated with software quality and reliability. The paper also examines what code coverage actually is and how the state-of-the-art suggests the usage of code coverage for optimal result. Through a Systematic Literature Review, with a holistic approach, the paper concludes that code coverage and software reliability are not directly correlated, but possibly indirectly with a varying correlation factor. When it comes to recommended code coverage practices and optimal use, the literature strongly argues for not having code coverage as an absolute goal of the testing, instead using it to evaluate the test suite and find its flaws so that it can be improved.

1 Introduction & Background

The world is becoming increasingly dependent on software due to the IT-revolution. Many products that traditionally were only mechanical now depend on an increasingly large degree of software. Software is becoming an integral part in a growing number of important functions in society today, controlling everything from toasters and fridges to important functions in cars, airplanes and huge safety-critical systems.

From a business perspective, it is imperative to ensure that the software cannot create liabilities for stakeholders. No matter if software is developed in-house or outsourced, software must meet the technical specification of the client where quality, reliability and cost is determined by the client prior to order. From a business and *Software Quality Management* (SQM) perspective, budgets should also be considered. As with research and development in general there are diminishing returns for testing and coverage after reaching their critical point in terms of efficiency. The higher the requirements for software reliability the more extensive and exhaustive the tests will need to be, which often generates higher costs.

Many factors are involved in the SQM processes. There is for, instance the budget to consider, system and software requirements, client specification, team and project organization as well as integrity and reliability of software. If bugs are found after release, the cost will be much higher to correct them compared to eliminating bugs during the early development process [Newman, 2002]. The adequacy of a testing process or test suite is increasingly important for the costs of the project as well as for the resulting software reliability. Testing is in many cases also an elimination of risks, which from a business perspective is sound as there are cases of external software and product audits, for example in the case of standards in aircrafts [Administration, 2003].

There are two different types of software verification: dynamic and static. *Static verification* is, in general, performed by not executing code and is often connected to a white-box approach in practice. *Dynamic Verification* is performed by executing code to a varying degree and is often connected to a black-box approach in practice. There are also practices like code inspection that are harder to categorize as static or dynamic, because of its non-technical nature. Code inspection is often just classified as a SQM practice, as its focus is often on improving quality attributes like readability and maintainability. However, it can often be used as Software Verification practice as well, in order to discover faults in the code. When it is included in Software Verification, it is usually categorized as a static practice, since the code is not run [Bourque et al., 2013]. *Code Coverage* is usually applied in white-box approaches, as it gives insight into the code for its analysis.

At the birth of the IT-industry, a low number of developers were needed for software development. A project could often involve only one or two developers. Following the rapid development, projects became increasingly large in terms of lines of code as well as in terms of the number of people involved. This has led to the need of larger test suites, need of more exhaustive tests, larger teams and more collaboration between teams in other offices, cities and other countries [Fischer, 2001].

As the IT-industry evolved throughout the last century, there was a need to rapidly evolve testing methods in order to ensure quality of software. The significance of delivering software with minimal errors has become more important over the years [White, 1987]. However, since the bugs are unknown until they are found, knowing the effectiveness of a testing process in catching all the bugs is inherently difficult. This is where *Coverage Testing* comes in, not as a means to test the software directly, but as a means of measuring the performed testing itself [Horgan et al., 1994]. With help of coverage analysis, of both the code and the test cases, the tester gets an idea of how much of the software has been executed and how much has not. All depending on what type of coverage the tester is interested in.

Coverage testing is not necessarily a guarantee of software quality or that the code that was run does not contain any bugs [Williams et al., 2001]. With knowledge about the software and its structure, coverage measures can give a good idea of how exhaustive testing of the software has been and indirectly connects coverage to software quality.

Testing coverage is increasingly relevant for its close connection to other related fields of fault detection and testing, in general, as is the case with automatic test case generation and model checking.

This paper will, through a *Systematic Literature Review* (SLR) [Kitchenham and Charters, 2007], condense and analyze how testing efforts based on coverage are related to the resulting software reliability and general software quality. It will also look into code coverage in general; its practical meaning and its interpretation. The research will use holistic approach and will examine what code coverage actually is [Booth et al., 2003], how it is or should be interpreted and the different ways in which coverage is used in the testing process.

By gathering the latest available research on the topic and analyzing the state-of-the-art, this paper will enable the parties of interest to understand how code coverage can affect the outcome of a software and its testing process. The information gathered can be of use by researchers and companies that are in need of an aid to understand the testing outcome. In addition, this paper will also suggest new angles, missing research and further studies for improvement.

1.1 Research Questions

- **RQ1** What is the state-of-the-art on the correlation between code coverage in software testing and software reliability and quality?
- **RQ2** What is code coverage, how is it interpreted and how should it be interpreted?
- **RQ3** Are there any combinations of testing practices, with at least one of them being a coverage practice, which are recommended for the Software Engineering industry as being more effective?

1.2 Paper Structure

This paper will have the following struture:

- Section 2 addresses the methodology that was used for this research; the established framework that it was based upon, the reasoning behind it and the possible threats to the validity of this research.
- Section 3 presents the results.
- Section 4 discusses relevant to the research questions and their relation to each other. This discussion will be based on the data that was gathered from the literature review.
- Section 5 concludes the research. The former section is rounded off with conclusions or new questions that justify further research in the field. Furthermore, the state-of-the-art is analyzed from the perspective of our topic and research questions, with suggestions for future research.

Furthermore, this research paper comes with a supplement document which contains the filled in Data Extraction Forms described in Data Extraction Strategy, Section 2.4, which connects strongly with both Section 2 and Section 3.

2 Methodology

This research project utilized a methodology that primarily followed Kitchenham's core framework for Systematic Literature Reviews [Kitchenham and Charters, 2007]. The research was customized to fit the authors' needs and requirements with emphasis on spent resources. Using a standardized method of analyzing input material for research, was deemed by the authors to generate higher validity over other literature selection and analysis methods. The primary benefits of utilizing the SLR approach were also that information over a wide array of primary and secondary sources was able to surface. Consistency in collected data would provide evidence to support an assumption. Inconsistency in collected data would provide bases of further research where the underlying principles of variation could be examined. All information was to be treated equally and this was judged by the authors to allow for an unbiased analysis, effectively increasing validity together with reliability of this study. This research did not utilize the aspect of quantitative meta-analysis in SLR suggested by Kitchenham [2004] due to the project resource limitations.

Another purpose of utilizing the SLR approach was to not only aggregate existing literature and primary information sources, but also to provide more evidence-based information as the cornerstone of this paper in order to increase validity. Identification of gaps in current research was deemed to become visible with this method. A complementary benefit of using this method was to provide a framework in which further research can be conducted.

2.1 Search Strategy

The aim for this step was to find as many primary studies as possible by using an unbiased search strategy [Kitchenham and Charters, 2007]. Five well-known and often used databases, specifically for computer sciences and software engineering, were included in order to uncover as many studies as possible. Most of the chosen databases showed peer-reviewed results only, which was an important quality aspect of the research. Table 1 presents the used databases.

Trial searches were executed for the search strategy in order to obtain mock results and a primary context of the real search result [Booth et al., 2003, Brereton et al., 2007]. Based off a few papers that were found during the trial searches, keywords were extracted for the search strategy. The searches were performed manually for each search term and the databases were traversed in the order in which they were listed as seen on Table 1. The only filtering and sorting criteria used for these searches were those presented in parentheses for each individual database in Table 1. Paired with the fact that the first 50 hits were considered, the search terms were designed to be fairly ambiguous. The authors chose to adapt to the holistic approach of the research in order to not exclude possibly important sources [Brereton et al., 2007].

The search terms that can be seen in Table 2 were found to be adequate.

2.2 Study Selection Criteria

This step was to determine which studies were included or excluded from the research by using criteria and a procedure. The criterion below were the necessary criterion for a paper to have the possibility of being included. Unless a paper met these criterias it was automatically excluded from the study. The procedure also described how selection criteria would be applied and how disagreements would be resolved [Kitchenham and Charters, 2007]. When it comes to the authors having to make subjective assessments, as described in inclusion criteria below, only two of the final papers went through this step after careful investigation.

Inclusion Criteria:

- Published within a 20 year timespan (1993-2013)
- Peer-reviewed and published papers only. In cases where doubts exists, the authors have the last word.
- Papers from the Software Engineering and Computer Science fields only.
- Papers are available in English.
- Full papers are available.

Selection Procedures:

Step 1 Each author individually selected relevant papers from the search hits according to the inclusion criteria described previously and any selected study goes through to Step 2.

Step 2 Each author read the introduction and conclusion of each paper and ranked them based on relevance (Yes, Maybe, No) in order to lower the amount of studies to the 25 most relevant. This final quantity of relevant papers was chosen arbitrarily based on the available resources for this research project, as well as the authors' estimation of cost in terms of time for the number of articles. Papers where both authors had given a "Yes" went straight to step 3 while papers with only one of the authors had given a "Yes" were discussed until a decision could be reached on whether to include the paper or not. If any openings among the final 25 papers still existed, the authors considered papers with two Maybes. If openings still remained, the authors closed the selection with the number of articles that had gone through at this point.

Step 3 Each author read the papers in its whole, took notes for internal use, and extracted data for each paper by filling out the data extraction form defined in Table 4.

2.3 Quality Assessment

This research project's main effort to do quality assessment on the selected studies was embedded in the inclusion criteria of the Study Selection criteria in Section 2.2. The demands for peer-review and publication on the papers, as well as the timespan of 20 years, kept the papers up-to-date and helped achieve a minimum quality for the selected papers. Since this research was considered to be an open-ended and holistic research project, the authors did not believe that it was constructive to judge papers based on their content in terms of ideas and reasoning for quality assessment.

Using a quality assessment ranking, as Kitchenham and Charters [2007] and Brereton et al. [2007] suggested, to differentiate the final papers which were selected based on quality was considered. However, Kitchenham was only dealing with one kind of study. This enabled her to use a quality ranking system such as DARE [Kitchenham and Charters, 2007]. While this study accepts all kinds of studies that contribute to the understanding of the topic and the research questions, it was not optimal to use such ranking system as this would require correct identification of different types of studies and implementation of different quality ranking system for each type of study. In the process of doing so, an additional avenue for human error that would

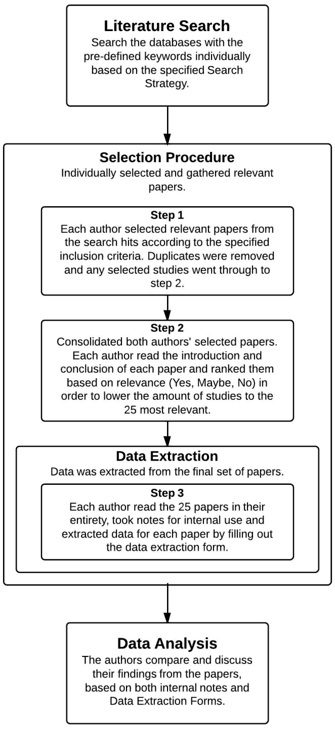
Literature Search
Search the databases with the pre-defined keywords individually based on the specified Search Strategy.
Selection Procedure
Individually selected and gathered relevant papers.
Step 1
Each author selected relevant papers from the search hits according to the specified inclusion criteria. Duplicates were removed and any selected studies went through to step 2.
Step 2
Consolidated both authors' selected papers. Each author read the introduction and conclusion of each paper and ranked them based on relevance (Yes, Maybe, No) in order to lower the amount of studies to the 25 most relevant.
Data Extraction
Data was extracted from the final set of papers.
Step 3
Each author read the 25 papers in their entirety, took notes for internal use and extracted data for each paper by filling out the data extraction form.
Data Analysis
The authors compare and discuss their findings from the papers, based on both internal notes and Data Extraction Forms.

Figure 1: The figure summarizes the applied methodology for this research.

Name	**URL**
IEEExplore	http://ieeexplore.ieee.org/
ACM Digital library	http://dl.acm.org
Google Scholar (No citations, no patents)	http://scholar.google.com
Citeseer Library	http://citeseer.ist.psu.edu
Springer Link (Articles, Computer Science, English; Do Not include preview-only content)	http://link.springer.com/

Table 1: Chosen databases

Search Term	**Synonyms**
Code Coverage	Test Coverage, Software Coverage
Coverage Testing	
Software Quality Coverage	Software Reliability Coverage

Table 2: Chosen Search Terms

threaten the validity of the research would be created. The conclusion was therefore not to apply this kind of a quality assessment method in this literature review. Having considered the resources allocated for this research project would be scarce, additional time constraints could not be implemented into the schedule and committed for just one part of the method applied.

2.4 Data Extraction Strategy

The process of data extraction and the data extraction form were based and designed on the guidelines for data extraction put forth in Kitchenham and Charters [2007] and was clear to both authors well in advance of this phase of the research, just as Brereton et al. [2007] suggested. Just as Kitchenham and Charters [2007] mentioned, the authors of this paper filtered down the guidelines and suggested steps to what was relevant for the topic, the research questions, the type of research, as well as the size of the research. The process was designed to include the following steps:

1. Both authors extract and fill in a data extraction form for each of the selected papers from the literature review.
2. Both agree or disagree on the information of the data forms.
3. Any disagreement is examined.
4. Record agreed final data.

The resulting template for the data extraction form can be seen in Table 4.

2.5 Statistics from Study Selection

The statistics from the specified Search Strategy and Study Selection Criteria, Sections 2.1 and 2.2 respectively, are presented in Table 3. The data was collected after the first and last step of the Study Selection procedure.

It was fairly clear from the numbers that there was a lot of overlap between the separate databases. Given that the authors actively rejected studies that had already been found, both during Step 1 and before Step 2 in Data Extraction Strategy, it can be clearly seen that latter databases had much more difficulty to contribute in terms of number of studies. Based on previous experiences with these databases, the authors had expected a database such as IEEExplore to initially give a considerable number of potentially relevant hits. The much larger amount of selected studies was heavily reflected by the fact that it was used before other databases and the remaining databases' lower numbers reflected on that as well.

Some databases proved, much according to the initial assumption of the authors, to be stronger or weaker than others on our topic. Citeseer Library only contributed three unique studies at the initial phase and at the end none of those had been chosen as part of the total 25 most important studies. While some databases did prove to be stronger or weaker than others in terms of selected studies, the search hit volumes presented in Table 3 should not necessarily be seen as each database's strength and potential, but rather as its capacity to contribute with something new on top of what all the previous databases contributed.

After Step 1 it could be seen that Google Scholar had contributed ten unique studies to the selection pool, a fair amount of studies considering it was being utilized after both IEEExplore and ACM Digital Library. SpringerLink performed slightly better than Google Scholar as the fifth and last database. Following Step 3 of the Study Selection procedure described in Section 2.2, search results showed that the quantity of papers found initially did not always correspond to an equivalent importance after filtration, as can be seen in Table 3. While IEEExplore had barely one sixth of the research papers found, Google Scholar had half of its initial studies and has more than twice as many as SpringerLink and almost as many as ACM Digital Library. This confirmed to the authors that including Google Scholar was a good choice, despite the initial notion of the authors to increase validity and reliability risks slightly.

2.6 Data Analysis

The first step was to read the papers individually, taking notes and filling out the data extraction forms specified in 2.4. The notes were part of the internal process followed by the authors and were not published in this paper unlike the data extraction forms. The reasoning behind the additional note-keeping was that the forms were good structure on what each individual paper contributed to each research question. However, they were inadequate for taking more general notes about the papers, demanding more extended notes useful for designing the discussion and conclusion parts of the paper. The authors then discussed and modified the notes in the data extraction form according to Section 2.4 Data Extraction Strategy . During this phase, there were discussions of what the collective trend of results were, what patterns and hypothesis, what conclusions could be made out of the data and what relevance the data had with the research questions as defined in Section 1.1.

2.7 Threats to Validity

From the initial stages of the methodology design to the completion of the project, the risks - or threats - to the validity of the paper's methods as well as its outcome were considered and discussed. This section presents what the authors believe to be the greatest of those risks. The presentation is based on the risk categories presented in Runeson and Höst [2009], meaning *Construct Validity*, *Internal Validity*, *External Validity* and *Reliability*. In the case of this research project and its perceived risks, only two of these categories were needed; Construct Validity and Reliability.

Database	Sel. Proc. – Step 1	Sel. Proc. Step 3
IEEExplore	68	11
ACM Digital Library	21	7
Google Scholar	10	5
Citeseer Library	3	0
Springer Link	14	2
Total	116	25

Table 3: Number of studies in contention after Step 1 and 3 of Selection Procedures

2.7.1 Construct Validity

By conducting this research based on a holistic approach additional risk was added to Section 2.1. The risk implied that the search strategy had to be ambiguous enough to encompass all possibly relevant data within the topic and at the same time, search terms could not be specific enough to guarantee that all highly relevant sources had been found within the number of search hits looked at. The authors had tried to counteract the increased risk factor by looking at the search hits and considering fifty of them for each search term in the specified databases. To the best knowledge of the authors, this number of considered search hits for each search term was to be considered a rather large size for the amount of resources available to this research project. While keeping in mind how the search hits for the different terms were expected to overlap considerately at that number of search hits, this research design was judged to counteract or even negate that risk.

2.7.2 Reliability

Considering how risks increase as soon as the human factor enters a process, there was always the risk that none of the authors identified a potentially interesting paper as described in Section 2.2 Study Selection Criteria. Another risk was that papers which were not selected could have been included in the twenty-five top papers. In order to counteract these risks the search and selection processes were based on individual work with a comparison and discussion at the end for the authors to reach a final agreement. However, it is difficult or even impossible, to negate the risks of a human error. Especially considering that they are counteracted by actions that include more human subjectivity.

Along the same lines, human errors comes with risk stemming from the use of Google Scholar as one of the selected databases. Since contents of Google Scholar do not include peer-reviewed and reliably published items only, the authors were bound to find cases that left some doubt on those points. Seeing as how those were important factors of the research's inclusion criteria for the Study Selection described in Section 2.2, as well as a fundamental part in the Quality Assessment presented in Section 2.3, that additional risk was something to take very seriously, due to its possible impact on the results and overall validity of this research. Therefore, in such cases, the authors agreed to base such decisions on publication source and citations in reliable sources. In any case where such a paper did not clearly contribute to the research, the paper was excluded. The authors believed this to be an acceptable trade-off since Google Scholar tends to give the user an exhaustive overview of available material across several sources, even from the less known databases. Fortunately, no such action was necessary during this procedure.

3 Results

This section presents the results from the Methodology up to Data Analysis as well as a short summary of the contents of the selected papers.This paper has been submitted with a separate *Supplementary Document* which contains all the filled out data extraction forms that are described in Section 2.4.

3.1 Results concerning Research Question 1

Most research on the topic concluded that there was some form of correlation between code coverage and software quality, albeit indirect in some cases [Malaiya et al., 2002, Veevers and Marshall, 1994, Chen et al., 1996, Piwowarski et al., 1993, Chen et al., 1997, Krishnamurthy and Mathur, 1996, Lyu, 2007]. Many researchers found that few studies were to be found regarding the fundamental relation between code coverage and quality (often measured in terms of failures and defects) [Mockus et al., 2009]. This conclusion was often based on software reliability being interpreted and incorporated as the main part of software quality. Many of the collected conclusions were based on intuition, empirical observations, isolated cases and very little in the form of facts and hard data, making them inconclusive Smith and Williams [2008] and Hamlet [1994]. Piwowarski et al. [1993] concluded that the correlation is in fact not direct, a case found by Krishnamurthy and Mathur [1996] as well. Furthermore, the correlation between code coverage and software quality did not work in one direction only. Testability can have a severe impact on coverage level achieved by a well-designed test suite Li [2005].

The correlation factor between code coverage and software reliability does in fact vary not only with the level of code coverage achieved, [Chen et al., 1996, Krishnamurthy and Mathur, 1996], or the type of code coverage criterion applied to a specific piece of code, but also with certain aspects of the code, like code complexity, code structure and its operational profile [Veevers and Marshall, 1994, Gupta and Jalote, 2008, Chen et al., 1996, Garg, 1994, Del Frate et al., 1995, Mockus et al., 2009]. Gupta and Jalote [2008] suggests that these factors should be considered when interpreting the results of the coverage analysis, something which is partly supported by Marick [1999].

According to data found, software quality and software reliability had different definitions depending on what study is examined. While Chen et al. [1996] and Lyu [2007] mentioned that reliability of a program is defined as probability that the program does not fail in a given environment during a given exposure time interval and that it is an important metric of software quality. Others such as Veevers and Marshall [1994] used reliability as a function of an idealized coverage metric. Many, if not most of the researchers, used code coverage as an indicator of software quality and reliability [Smith and Williams, 2008, Horgan et al., 1994, Gittens et al., 2006].

One popular definition of reliability was to use fault detection as a basis [Horgan et al., 1994, Chen et al., 1996, Piwowarski et al., 1993], but it was not the only definition [Del Frate et al., 1995, Lyu, 2007]. Garg [1994] actually separated the study's implicit definition of reliability from fault detection, and found that code coverage had a greater correlation with this definition of reliability than it had with fault detection, something that Lyu [2007] had found as well. In fact, an increase in fault detection did not seem to guarantee an increase in neither reliability nor code coverage [Krishnamurthy and Mathur, 1996, Del Frate et al., 1995]. Cai and Lyu [2005] even claimed that there was no direct correlation between fault detection and code coverage at all.

However, Namin and Andrews [2009] found that the correlation between code coverage and fault detection increased to a certain degree when test suite minimization was performed on the test suite, keeping the number of test cases to a minimum for the coverage level. Some researchers had even found that by increasing fault detection, improving testability and test suites [Horgan et al., 1994, Marick, 1999, Li, 2005, Chen et al., 1996], reliability tended to increase. Chen et al. [1996] found that the reliability of a software only increased when a number of faults were removed from a software. Horgan et al. [1994] stated on the other hand that a high level of coverage (between 80-90% coverage) was a crucial step towards software quality and reliability. This was supported by Barnes and Hopkins [2007], indicating that software quality could not be trusted if coverage was low, and by Mockus et al. [2009], mentioning that an increase in coverage tended to result in an increased reliability. Many results found by Lyu [2007] concluded that high code coverage tends to add high software reliability and an increase in reliability comes with an increase in at least one code coverage measure.

However, Horgan et al. [1994] also saw that there was a correlation between the number of faults detected during testing of a version and coverage of a program. Horgan et al. [1994] did not see the correlation during various testing conditions due to the number of faults detected not being sufficient enough to draw such conclusion. Cai and Lyu [2005] showed on the other hand that the effect of code coverage on fault detection varies depending on the testing profile and that the correlation is high when it comes to exceptional test cases and low when it comes to normal testing. Malaiya et al. [2002] highlighted that software reliability is affected by factors such as testing strategy, the relationship between calendar time and execution time and testing of rarely executed modules by creating a model. Their research also states that even though one may reach 100% code coverage it is not necessary that all faults have been found. Lyu [2007] added that the knowledge and training a software engineers also affected software reliability as practitioners tend to perceive software reliability analysis as a non-productive task and are often suspicious about the reliability numbers obtained by the available reliability models. Since those numbers, as reliability achievements, obtained from different projects could rarely be validated.

This research project also found that there has been considerable research in growth models [Chen et al., 1997, 1996, Veevers and Marshall, 1994, Inoue and Yamada, 2004]. One type of growth models that have been given a lot of attention are the *Software Reliability Growth Models* (SRGMs), which attempt to predict the evolution of software reliability for software under testing. Chen et al. [1997] enforced the idea that there is a correlation between software reliability and code coverage by showing that a SRGM approach based on both time and coverage not only produced more accurate predictions, but also mitigated the general SRGM tendency to make overoptimistic predictions. This disadvantage in more general SRGMs is due to their omission of a testing method's natural limitations as the code coverage level increases to higher levels. However, Li [2005] voiced concerns about the hurdles in getting SRGMs used in industry settings, starting with the sheer number of existing models that have all justified their existence in one way or another, and without any indication of which one to choose. In some cases however, these growth models are claimed to be adapted for easy understanding and usage, like the model presented in Malaiya et al. [2002].

3.2 Results concerning Research Question 2

Smith and Williams [2008] refers to previous work to describe code coverage indirectly by indicating that ”there should be no possibility that an unusual combination of input data or conditions may bring to light an unexpected mistake in the program”. Lyu [2007] described code coverage as an indication of how completely a test set executes a software system under test, therefore influencing the resulting reliability measure, and Cai and Lyu [2005] proposed code coverage an indicator of testing effectiveness and completeness. However, Namin and Andrews [2009] points out that it is not always true that code coverage is a good indicator of effectiveness, and it is generally only true when size is controlled for.

Marick [1999] states a strong belief in that code coverage analysis in software testing is of great value and can help practitioners improve the resulting software quality. As was stated in the paper ”they’re [coverage tools] only helpful if they’re used to enhance thought, not replace it.”. However, Marick [1999] insists that code coverage is applied and interpreted incorrectly more than it is not. This incorrect usage of code coverage stems to a large degree from insistence of having coverage results be a direct indicator of software reliability and quality and making the coverage numbers a main goal of the test suite design process. Marick [1999] instead means that code coverage does not serve as an absolute ruling of quality, but as a guideline that somebody with domain experience should be able to interpret as an adequacy criteria for the testing, as well as use the data to improve general test suite design if any flaws exist. In essence, test suite design should be done without focusing on coverage goals, and the results from the coverage analysis should reflect on the test suite adequacy and not on software quality per se, according to Marick [1999]. Chen et al. [1996] supports this line of thought, stating that coverage data from the testing process should be used as a guideline for the testing process’ general effectiveness and efficiency and not as an absolute ruling of quality. Meanwhile, Kessis et al. [2005] refers to Marick [1999] as well when stating that coverage analysis is often done wrong, having the coverage numbers be the absolute goal of the testing process in respect to code coverage. Furthermore, both Williams et al. [2001] and Barnes and Hopkins [2007] also reason along the same lines, stating that 100% coverage on its own is not any kind of guarantee and does not tell the tester anything about the code. It is suggested that even having two test suites, the same coverage metric and the same coverage level, it does still not mean that the test suites are equivalent, neither in effectiveness nor in efficiency.

Possibly being connected to the interpretation and usage suggestions by mainly Marick [1999], Hamlet [1994] presents a theory proposing that reliability or ”trustworthiness” actually consists of two separate parts; the technical part that could potentially be measured based on e.g. *mean time to failure* (MTTF) or fault detection and the second part which is called ”dependability” in Hamlet [1994], which is the part that corresponds to the intuitive property of ”trustworthiness”, the part that is hard to measure and open to interpretation and estimation, although its clear dependency on the technical part.

3.3 Results concerning Research Question 3

The results have shown that there is not one particular practice that is being considered substantially superior when it comes to testing. Since each testing practice had been tailored for each research, they all had their own pros and cons. A research by Gupta and Jalote [2008] developed a tool which the group used to look into each code coverage type and used mutation analysis to see the effectiveness and efficiency of each type. During the research, the group found that predicate coverage showed the best effectiveness but required more testing effort than block coverage and branch coverage. While block coverage took least testing effort at a lower effectiveness, it was found that it had higher efficiency but with greater variability, possibly making it ill-suited to measure the reliability of a program. Barnes and Hopkins [2007] developed a framework of routines, in terms of reduced collection of datasets, for the LAPACK suite. The test suite minimization allowed the researchers to greatly lower the cost, in term of CPU-power and time, when the suites were executed. Another research Chen et al. [1996] presented a technique which models the failure rate with respect to both test coverage and testing time. Such failure rate was applied to a software reliability growth model (SRGM) and reliability overestimation was observed. The new approach by Chen et al. [1996] helped the SRGM make more accurate predictions, as well as reveal the efficiency of a testing profile which helps the developers conduct a more effective testing scheme. In other words, it improved the applicability and performance of the SRGM. Namin and Andrews [2009] on the other hand found that coverage could be correlated with effectiveness when size is controlled. By randomly generating test suites and analyzing plotted correlation between coverage and effectiveness as the size was held consistent, Namin and Andrews [2009] could then see if there were any consistent patterns. The result was not very clear but many test suites indicated that the correlation was high and seldom close to zero.

Gittens et al. [2006] suggests that the correlation between code coverage and software quality starts weakening after reaching 70% code coverage, while Horgan et al. [1994] concluded that 80%–90% code coverage had shown to be crucial in order to achieve software reliability and software quality in general. However, work has been done that claims no diminishing returns past 80% code coverage, though the cost tends to start increasing exponentially [Mockus et al., 2009]. Piwowarski et al. [1993] concluded that 70-80% is the critical point for code coverage in terms of cost efficiency, even when the coverage metric used is statement coverage. Using and trusting in relatively weak coverage metrics like statement, line- or even block coverage is something that Malaiya et al. [1994] warns about, instead suggesting that 80% branch coverage is often adequate. Kessis et al. [2005] further states that the larger and more complex the code base is, the more infeasible it is to achieve code coverage levels around 100%. However, Marick [1999] goes on and suggests designing test suites aiming for 100% coverage is not only extremely cost ineffective, but also waters down the usefulness of the coverage analysis in order to achieve it, making the test suite uniformly weak at finding flaws and omissions in the testing process.

Because of cost efficiency concerns, practitioners sometimes do not strive for high code coverage, since coverage analysis is still often seen as an indirect, non-productive task in the projects, with no immediate returns [Kessis et al., 2005]. Piwowarski et al. [1993]’s data does suggest though, that up until reaching the critical point for code coverage, it is in general worth the resources spent improving code coverage. It has also been suggested that testers should consider using combinations of different coverage criterion, in order to more easily find different types of faults [Kessis et al., 2005]. However, Marick [1999] does point out that coverage analysis is more useful to developer testers than it is for general testers, since the latter have objectives more based on requirements-level or specification-level omissions and this type of errors would be caught less effectively by coverage analysis, if they are caught at all.

The Smith and Williams [2008] survey also shows that even in the industry, many practitioners do not use code coverage as a way of measuring testing adequacy in unit testing, even though a considerable part of them use code coverage to some extent in testing. Some simply use it because it is obligatory, others use it out of habit and indoctrination without having considered its meaning for the testing and some actually use it with clear intent, just not as a stopping criterion in testing.

Code prioritization techniques seem to be among the most popular approaches and practices in order to try to optimize test results with code coverage properties [Li, 2005]. Marick [1999] used code prioritization in the step-by-step test suite design approach and Kim [2003] uses it as well in the proposed coverage analysis method. Kim [2003]'s proposal consists of assigning coarse or detailed coverage testing requirements to modules depending on their proneness to contain faults in the past, optimizing the effectiveness and efficiency of the testing process based on the statement that 70% of bugs come from only 20% of modules. It has also been interpreted from data that code coverage may be more efficient in finding faults in exceptional test cases than in normal test cases. However, it requires domain knowledge and code prioritization experience. The better this is input to the test suite design is, the better the potential effects of it on the testing process and on the software. However, the danger in this is that the opposite also applies [Cai and Lyu, 2005].

Another fairly popular approach concerning code coverage in the testing process is test suite minimization, as used by e.g. Barnes and Hopkins [2007] . Test suite minimization is done to improve the efficiency of a test suite without giving up much of its original effectiveness, effectiveness that is measured by code coverage.

4 Discussion

This Section discusses the results from the Literature Search and Selection Procedure, described in Sections 2.1, 2.2 and in Table 1, including some statistics. It also discusses the contents of the state-of-the-art on these topics, the selected papers, if there are any patterns in the set of papers and how they relate to each other.

4.1 Discussion concerning Research Question 1

In the discussions conducted in the literature concerning the topic of Research Question 1, as defined in Section 1.1, quite a few assumptions are made, with special emphasis on those that conclude that there is some kind of correlation between code coverage and software reliability [Malaiya et al., 2002, Veevers and Marshall, 1994, Chen et al., 1996, Piwowarski et al., 1993, Chen et al., 1997, Krishnamurthy and Mathur, 1996, Lyu, 2007]. However, Mockus et al. [2009] pointed out that many researchers have observed that there is little to no actual proof to support a direct correlation. In fact, many have found that the correlation factor between the two varies by quite a bit, being influenced by factors like e.g. code complexity, code structure and its operational profile [Veevers and Marshall, 1994, Gupta and Jalote, 2008, Chen et al., 1996, Garg, 1994, Del Frate et al., 1995, Mockus et al., 2009], among other factors.

Many of these conclusions or assumptions about existence of a direct correlation are based on intuition and empirical observations, making them very inconclusive, as stated in Smith and Williams [2008] and Hamlet [1994]. Still, research based on this unproven premise continues to be performed. However, the authors of this paper agree that some form of varying, indirect correlation between code coverage and software reliability seems to exist.

Another area which has received a lot of attention is that of growth models [Chen et al., 1997, 1996, Veevers and Marshall, 1994, Inoue and Yamada, 2004], more specifically SRGMs. The research into SRGMs has reached the same conclusion of the previous papers ([Malaiya et al., 2002, Veevers and Marshall, 1994, Chen et al., 1996, Piwowarski et al., 1993, Chen et al., 1997, Krishnamurthy and Mathur, 1996, Lyu, 2007]) , as models based on both time and code coverage have been able to make more accurate estimations and avoid the tendency that SRGM's not based on coverage usually have tendencies of being overoptimistic.

However, the authors of this paper found it surprising to discover while reading the literature that such a basic building block in this topic, and in fact in the definition of Software Quality in general, as the term ”reliability” seems to not be clearly defined between the different papers. Some didn't bother defining it and probably assumed that the term ”reliability” was clear enough, some defined it based on fault detection Horgan et al. [1994], Chen et al. [1996], Piwowarski et al. [1993] and others on MTTF [Del Frate et al., 1995, Lyu, 2007]. While both those definitions can be argued to be close to the traditional definition of reliability, it is still cause for confusion, as everyone is not writing about the same reliability. The question is what legitimacy one can assume that this work has, because while it does not invalidate the work done within the actual research projects, if the premise is wrong, the conclusion may be wrong. The authors of this paper believe that such an important term in *Software Engineering* should not be defined differently from one paper to another and so often that such a pattern is easily observable. Considering that there is an ISO standard for software quality [ISO, 2013] as well as the IEEE definition of software quality [IES, 1998] and SQM best practice, the authors of the collected data should have clearly referred their definition of quality and reliability to the standards as it otherwise is a rather ambiguous term and can cause a lot of confusion. The actual definition of reliability thus must be discussed due to the large amount of research concerning software reliability and software quality. It is therefore essential that studies are generally referring to the same definition of “reliability” and “quality”. Defining software reliability and software quality in general is however beyond the scope of this study.

Since software reliability was often based on fault detection in the papers included in this research project, the authors found it interesting that Garg [1994] and Lyu [2007] actually found that code coverage as a greater correlation with what Garg [1994] called ”true” reliability than what ”true” reliability had with fault detection. Other research papers have supported these findings as well [Krishnamurthy and Mathur, 1996, Del Frate et al., 1995], and one even states that there is no correlation between code coverage and fault detection [Cai and Lyu, 2005]. This poses another interesting question in respect to the research done and the results obtained where software reliability has been based on fault detection: Have these research papers even been using software reliability? This situation is quite unexpected considering the number of papers that it could affect the results of.

4.2 Discussion concerning Research Question 2

Before this discussion addresses how code coverage is being used and how it should supposedly be used and interpreted, it’s important to discuss what code coverage is. Fortunately, the definition of code coverage seems to be fairly straightforward and there seems to be consensus as far as the selected research papers go, unlike the definition of software reliability which was presented in Results concerning Research Question 1 and discussed in Discussion of Research Question 1, Sections 3.1 and 4.1 respectively. While not many give it enough thought to describe it, and those that do sometimes rather relate it to the code coverage goals [Cai and Lyu, 2005] or describe the reason for performing thorough testing [Smith and Williams, 2008], there are those that describe it more traditionally as a way of evaluating the thoroughness and completeness of testing [Lyu, 2007]. The authors of this paper have also noticed that in the reasoning, it is the latter description that researchers seem to use, as a way of specifying what has been covered at least once by testing and what has not. In the search for software engineering practices that would improve and assess software quality in a product before release, using code coverage in testing becomes an intuitively justified choice. In essence, the main argument for coverage analysis has always been not what it says about the code that is run during testing, but what it would not say about the testing if code coverage was not analyzed [Hamlet, 1994].

When it comes to how code coverage is used and interpreted, it was presented in Results concerning Research Question 2, Section 3.2, how claims are stated in Marick [1999] of how code coverage is often used and interpreted incorrectly in practice. Marick [1999] even gives some very good examples, counter-proposals, and reasoning about usual mistakes that are done with code coverage. Essentially, the argumentation and reasoning in Marick [1999] is connected to this paper’s first research question, what the correlation is between code coverage, software reliability and software quality in general. The interpretation made by the authors of this research paper is that this argumentation in Marick [1999] follows the line of thought that there is no direct correlation between code coverage and software reliability. While many studies have concluded that there is indirect and varying correlation between code coverage and reliability, it might be because of this oversimplification and insistence that says that code coverage is correlated with reliability, that code coverage is used in what Marick [1999], with support from Chen et al. [1996], Kessis et al. [2005] and partially from Williams et al. [2001], Barnes and Hopkins [2007], states are incorrect ways.

What this paper’s authors found most interesting in this case was that it wasn’t just in the industry that this insistence was done. In fact, excluding the papers [Marick, 1999, Chen et al., 1996, Kessis et al., 2005] that actually brought up this question, many of the selected papers seemed to approach the subject of the correlation between code coverage and software reliability, as well as the usage and interpretation of code coverage in software testing, the exact way that was argued against in Marick [1999]. One could argue that it is being done for academic purposes in some cases, which may be true, but in some papers more than others it’s quite clear that code coverage is being seen as a direct indicator of reliability or effectiveness (maybe add reference).

Despite the claimed misuse of code coverage, Marick [1999] expresses strong belief in code coverage and the power of coverage analysis in software testing. Instead of using code coverage as an indicator of reliability (or quality) and make it into the absolute goal, it is suggested that code coverage should be used for evaluating and analyzing the test suite, in order to interpret the results with domain knowledge and evaluate the test suite. With other words, code coverage relates directly to test suite design, which in turn may be the indirect link between code coverage and software quality. By having a well-designed test suite for the purposes and goals of the testing activities, one may well achieve the quality and reliability goals that were being aimed for. Marick [1999] did not explicitly define what a well-designed test suite is and even stated that it would be highly influenced by the code itself, requiring domain knowledge. However, it can be interpreted from its discussion and argumentation that a well-designed test suite should aspire to cover all the major parts of the code's structure, adding coverage of features, coverage of code segments with higher complexity or fault probability and tests of recoverable errors to that. A well-designed test should not include test cases which only exist to increase code coverage. Marick's own approach also suggests using a risk-based code prioritization technique in order to focus the project's resources where they are of most use.

Another interesting branch of this discussion is the observation in Li [2005], presented in Results concerning Research Question 1, Section 3.1, claiming that while code coverage may only be indirectly correlated with reliability and quality, the code coverage level that a well-designed test suite can achieve is severely bottlenecked by the software's testability attribute. It is often hard to satisfy high code coverage goals of a software with low testability, even if the test suite is well-designed. Naturally, like when code coverage is an absolute goal, it is possible to reach the level one wants by adding more test cases which increase the coverage level. However, as this paper presented in Results concerning Research Question 1, Section 3.1, Marick [1999] claimed that this kind of approach to test suite effectiveness only waters down the capabilities of the coverage analysis of finding flaws in the test suite design, as the test suite will become uniformly weak and be unable to give any hints about which parts that needs to be improved.

Finally, and keeping the previous discussion in mind, these aspects and interpretations of code coverage use may be connected to the discussion in Hamlet [1994] of two separate parts of what people intuitively define as the term reliability, or ”trustworthiness”. In essence, the reasoning in Hamlet [1994] divided reliability into a technical part, this is the part that can usually be measured based on e.g. MTTF or fault detection, and the intuitive part which was called ”dependability” and which is open for estimation and interpretation of objective results from the technical parts on which it is dependent. The latter part can usually not be objectively measured, without taking utilizing measurements from the technical part. While the results and numbers from the technical part may be objective, just as is the case with other objective numbers, they may be misinterpreted. This is their danger, when they're misinterpreted.

It is the belief of this research paper's authors that practitioners often define reliability as the second part of the definition in Hamlet [1994], the intuitive part. As the nature of estimating or testing a software for reliability makes it practically impossible to come up with definitive results of the reliability, as the measuring methods do not know when adequacy or maximum level is reached, this turns the attempts of measuring reliability and quality into code coverage measurements related to e.g. MTTF and fault detection, with other words, the first part of the definition. Having made such a decision will still not be able to guarantee that the results are definitive either, but at least here objective numbers and results are obtained. This may have led parts of the field into eagerly trying to correlate code coverage directly with reliability, in order to reach some results in their search for an objective result for reliability, even though research seems to have found little to no hard facts confirming that they're directly correlated [Smith and Williams, 2008, Hamlet, 1994]. In fact, as was presented in Results concerning Research Question 1, Section 3.1, most research either concludes or finds strong indications that the correlation between code coverage and reliability is varying and indirect [Malaiya et al., 2002, Veevers and Marshall, 1994, Chen et al., 1996, Piwowarski et al., 1993, Chen et al., 1997, Krishnamurthy and Mathur, 1996, Lyu, 2007].

Based on the discussion in Hamlet [1994] about the definition of ”dependability” within reliability, one has to consider if this leads to the discussion in Marick [1999], about how code coverage should be used for test suite adequacy and improvement. Perhaps the use of non-definitive measurements from within the technical part of the definition of reliability in Hamlet [1994] and using it as a definitive result of ”dependability” is what created the insistence in both the industry and in parts of the academic world that code coverage and software reliability are directly correlated, even though no proof of that exists, as was stated previously in the discussion.

4.3 Discussion concerning Research Question 3

This SLR has found that the state-of-the-art research on the topic of research question 3, as defined in Section 1.1 does not seem to have any clear suggestion for testing practices that also include code coverage practices that are recommended as being more effective. While it is true that this was the research question out of 3 that this study focused less on, the authors of this paper still believe that by being so close to the other 2 research questions and by applying such an wide search strategy, as was described in Section Search Strategy, 2.1, there is no reason to believe that they were missed by the literature search. In fact, after having researched the previous 2 research questions it seems fairly natural that this is the case. As the most effective testing practices already vary a lot depending on the software that is to be tested, adding in the code coverage aspect probably increases the variability even more, as we have presented in Results concerning Research Question 1 and Discussion about Research Question 1, Sections 3.1 and 3.2 respectively, that there are many important factors that can change the correlation factor between code coverage and fault detection, effectiveness, reliability and even code coverage in general.

As for the patterns that were observed in relation to research question 3, as defined in Section 1.1, based on the 25 papers that were included in this research paper, code prioritization was the most popular suggested practice to perform in unison with code coverage practices in the testing process [Li, 2005, Marick, 1999, Kim, 2003]. This approach was used both to increase effectiveness of the test suite and increase its efficiency as well. Kim [2003] added that in general, 70% of the faults exist in 20% of the modules. This means that with adequate code prioritization one should be able to increase testing effectiveness in regard to fault detection by focusing more resources on the error-prone modules and increase efficiency by focusing less on modules that are less error-prone. It should be mentioned that this study was focused on large-scale industrial projects and clearly states that its results do not claim that the suggested approach would work on software projects of any size. The disadvantage of these approaches as far as recommending it as part of a systematic testing method is that code prioritization usually requires considerable domain knowledge and experience. The more the better. The results and effect on the software and the testing process will always greatly depend on this input to the test suite design, for better and for worse results. This adds quite a lot of requirements, subjectivity, and potential risk to a very central part of the testing process, making it a very risky general practice recommendation. However, with the right people involved, the authors of this paper believe it could have very good results in achieving its general goals.

Another proposed testing practice to use in unison with code coverage was test suite minimization. This testing practice was mostly proposed in conjunction with large-scale, complex software systems Barnes and Hopkins [2007], where increased efficiency without sacrificing effectiveness is of high value. It is also proposed for academic purposes, e.g. when one is trying to measure the correlation between code coverage and fault detection [Garg, 1994, Lyu, 2007]. For this latter purpose it works well, however the interesting thing is that it was argued in Namin and Andrews [2009] that test cases that don't add any additional coverage to the test suite and just run part of the code in a different way, may manipulate the correlation results by finding additional bugs that the first run through that code piece did not find. Keeping in mind the discussion about correct use and interpretation of code coverage practices in Marick [1999] which was discussed in Discussion of Research Question 2, Section 4.2, one should not have code coverage as an absolute goal for the test suite. The main goal should be having a good test suite and test process, which is more likely to be achieved if one focuses on designing a good test suite for the software at hand. If there are test cases that have proven themselves and their ability to find unique faults, why then remove those test cases? Intuitively, unless the cost in terms of lower efficiency is so high that the potential fault detection ability of these test cases makes them too expensive to be worthwhile, it seems counterproductive to exclude them from the test case according to this paper's authors. In fact, what is happening in such a situation is probably that by giving the test suite these redundant test cases, the test suite designers have unwittingly given it the capability of partly performing testing according to another coverage criteria and perhaps even a more thorough one. In fact, trying different combinations of coverage criterion is even something that Kessis et al. [2005] suggested in order to more easily find different types of faults.

However, because of cost efficiency concerns, practitioners sometimes see code coverage or the testing phase in general as a time consuming, non-productive task with no imediate returns [Kessis et al., 2005], even though the data in Piwowarski et al. [1993] seems to suggest that until reaching the critical point for code coverage in terms of its trade-off between effectiveness and efficiency, it is generally worth the resources spent on improving testing results and code coverage measurements.

Some data was found concerning how the different coverage criterion generally compare, excluding all the other possible factors that were mentioned at the start of this discussion. Several suggestions about the coverage criteria and coverage level that different researchers believe to be optimal in general have been suggested. Note however, that in Malaiya et al. [1994] a warning is put forth concerning the use of the weaker coverage criterion, like statement-, line- and even block coverage.

In terms of coverage level suggested, it generally ranges between 70% and 90% with varying criterion. Piwowarski et al. [1993] suggested that 80% branch coverage is generally enough, unless one knows that the software is very reliability-dependent and one needs to aim for higher reliability assurance than just reaching the critical point in coverage analysis between cost and effectiveness. These general recommended coverage levels are somewhere around what the authors of this paper imagined they would find, after consulting industry contacts prior to conducting this study.

In terms of criterion, it ranges from statement- and line coverage, through block coverage and up to branch- and predicate coverage. Other popular criterion used in research are MC/DC coverage, dataflow criteria like c-use and p-use, and even up to the level of path coverage. Of course, the more thorough the criterion and the bigger size and more complex the software, the faster these more expensive criteria get out of hand in terms of cost scalability. However, it was curious to notice, not only in the papers used in this research but in many more papers that were looked at during the selection procedure described in either Section 2.2 or Table 1, how these selected criteria have changed throughout the years. The authors of this paper consider if, perhaps in the future, hardware performance, coverage criterion theory and algorithms will have improved to the point where a coverage criteria like path coverage may be perfectly feasible to perform even in large-scale, complex software systems, just as block- and branch coverage have become. As path coverage is considered to subsume all the previously mentioned criterion, that may be the point where a clear answer can finally be given on which coverage criteria to use.

5 Conclusions

This research paper applied a (SLR) where it took a holistic approach to explore code coverage and its correlation with software reliability. Furthermore, as a literature review, it should analyze and summarize the state-of-the-art on these topics and as a study made with a holistic approach it is the actual exploration and discussion of the topic that is the main contribution. Nevertheless, the authors did naturally arrive at some conclusions and thoughts stemming from the Discussion (Section 4) as well. Since the presentation and discussion of the topics has been divided by research question, this section will follow that same layout.

- **RQ1** What is the state-of-the-art research on the correlation between code coverage in software testing, software reliability and software quality in general?

From the discussion, the authors can conclude that the selected papers show that there is no direct correlation between code coverage and software reliability. On the other hand, there seems to be very strong indication that there is an indirect correlation between the two as a relatively large amount of the selected papers have reached this conclusion. Additionally, even SRGMs have become better when including code coverage data in their calculations. However, many are still basing this belief on intuition and base their research on it without trying to find proof of such a correlation first, as if it was already concluded. If that premise does not hold, their results may not hold either.

Apart from the discussion of whether the observed relation between code coverage and software reliability implies that the correlation is direct or indirect, the main discussion on this question was actually the basis used for some of these studies. On the correlation question above there were unclear premises, but the authors also discovered that even such a basic building block in any research or discussion about Software Quality as Software Reliability should be clear and the same from paper to paper, there are after all many standards on Software Quality, there is no need to be ambiguous. As it was mentioned in the discussion, it could be argued that MTTF and fault detection are still fairly close to the traditional definition of software reliability and just a way of measuring it, but it still remains a potential source of confusion and invalidity of research if researchers are using the same terminology but defining it differently. In such a case, research papers could simply call it MTTF or fault detection as they will need to specify it anyway. This potential nonalignment between the selected papers becomes even more apparent when there was actually research stating that code coverage had higher correlation with reliability than fault detection had with any of them.

- **RQ2** What is code coverage, how is it interpreted and how should it be interpreted?

The discussion, thoughts and conclusions on this question revolve a lot around the suggestions and interpretations of code coverage and its use. There are some very good arguments and reasoning in the literature about how to use and not to use code coverage in testing that actually goes against how code coverage is often used, in the industry as well as by some researchers. Essentially, practitioners should not design test suites for code coverage and make code coverage an absolute goal of its own, instead they should focus on designing as good test suites as they can. Coverage analysis of testing can give indications and hints about any potential flaws in the test suite, enabling the practitioners to improve it. If the test suite is well-designed for the software under test, code coverage tends to come on its own, at least if the software's testability permits the coverage level to reach the goal.

There may be something else that is often interpreted into the term reliability that one of the papers calls ”dependability”. ”Dependability” is, in essence, the part of reliability that is based on intuition and estimation, making it practically impossible to objectively measure. The authors of this paper concluded it might hold some truth to it, as it could explain why intuitively people have a tendency to correlate code coverage with reliability, when in reality what is being measured is the technical part that is measuring something else.

- **RQ3** Are there any combinations of testing practices, with at least one of them being a coverage practice, which are recommended for the Software Engineering industry as being more effective?

There are not any recommended testing practices that in general are considered more effective than the rest. After researching the other two research questions, the authors of this paper believe the reason is fairly clear. The best testing practices already vary with the software and factors like complexity, code structure, size, so it is hard to recommend any specific set of practices in general. By adding code coverage practices into the mix even more variability is added to the decision.

Based on the selected papers in this research, the most popular practice to research seemed to be code prioritization, as a means of increasing both effectiveness and efficiency. This may sound like a good general recommendation, but as is discussed in the paper it comes with its downsides in the form of increased requirements, increased variation in the results and increased subjective decisions in the testing process that heavily influences the results. For a general testing practice, the authors of this paper do not believe it can be recommended.

There were some suggestions of recommended coverage levels and criteria in the papers, the coverage level generally ranged between 70% and 90%, while the variations in criterion varies a little more, with the strongest criterion being predicate coverage in those suggestions. Just like the testing practices, such a recommendation would vary a lot with the focus on reliability, code complexity, code structure and more. It is very difficult to give a general recommendation as *more* effective. The authors of this paper consider if the technological advances, in form of hardware performance, coverage theory and algorithms, will not make the current recommendations completely obsolete in the close future. As stronger criterion, e.g. path coverage, that subsumes all the previous will become feasible to perform even on larger or more complex code, just as the recommended coverage criterion have been changing up until now and criterion that used to be infeasible suddenly are not. That may be the point where there will be one clear answer to this question when it comes to coverage criterion.

- **State-of-the-art**

When it comes to the state-of-the-art on these topics, the current situation has partly been described in the conclusion for the questions. On the correlation between code coverage and software reliability a lot of interesting work has been done, and from many different perspectives on top of that. What is missing is mostly a definitive conclusion to the question. The closest thing to that was the conclusion that the correlation between them was not direct, but there's no definitive conclusion as to what it is then. Then of course the reliability term situation. The authors of this paper are convinced that having researchers agree to use the same definition for such a basic building block as reliability within software quality would immediately improve the chances of significant advances on this topic.

On the topic of code coverage and its use, there wasn't too much to find. While it's true that Marick [1999] made a very good case for his perspective of on the topic and some supported his positions on it, there was not much more to add to that. The fact that Hamlet [1994] had some very interesting ideas that connected very well with the opinions on code coverage in Marick [1999]. However, that was just a coincidence as the Hamlet [1994] paper was actually about reliability definition and not on code coverage. Marick [1999] is becoming slightly aged, while still very relevant, it would be good to get some new ideas into this topic with ideas from the present state of code coverage testing.

The recommended testing practices including code coverage practices didn't really exist. However, as was mentioned in the conclusion for Research Question 3, there is probably a good reason for it, as it is very difficult to recommend general testing practices that should be more effective than most. Rather the testing practices should probably be adapted to the software for optimal results. The authors of this paper are slightly surprised that there was not more material on recommended coverage criterion and levels for different types of software, and discussions that compare the different criterions against each other.

Finally, the authors of this paper can also add that there were indications that the Literature Search and Selection Procedures worked well in gathering the state-of-the-art literature on the topics. In several of our topics, like SRGMs, reliability correlation or code coverage definition and use, there was considerable cross-referencing between the papers. This reinforces the authors' belief that the methods used were adequate and managed to do a thorough job collecting papers, while the authors' relevance ranking as a final filtration step also worked fairly well.

6 Future Work

First and foremost, the authors strongly believe that a stricter definition of what reliability is and how it is measured would greatly improve the chances of considerable advances on not only the topic of correlation between reliability and code coverage, but other topics based on software reliability. In fact, it would be a good idea to incorporate and consider the Hamlet [1994] paper's ideas on software reliability definition. One of the main strengths of the research world is that researchers build on each others work to advance mankind's knowledge. By having such a basic building block like software reliability open for different definitions, it causes confusion as to if different papers are doing directly comparable research or not and getting comparable results or not.

The authors also suggest an extensive primary study of correlation between code coverage and software reliability to be made, the aim should be to conclude or reject the idea that there is an indirect correlation between code coverage and software reliability. For example, a worldwide or country wide survey could be issued to businesses in the IT-industry in order to secure current best practice. Another method would be to perform a large scale quantitative study of research and literature, encompassing as much as possible of current research.

There had been a lot of research done on growth models and especially SRGMs, however the hurdle for adopting an SRGM in the industry is very big. The authors of this paper believe that the biggest hurdle by far is just the sheer number of currently existing models, all of them justifying their existence with at least one isolated case. The field would do well with some convergence in order to greatly diminish the number of existing models, or possibly conclude some of them that generally work reasonably well.

Seeing that many researchers have different definitions of software reliability how closely connected software reliability is to software quality, the research area could benefit from identifying, analyzing and suggesting standardized definitions of Software reliability or Software quality.

Acknowledgements

The authors of this paper would like to thank all the teachers at the University of Gothenburg's Software Engineering and Management bachelor programme who have supported and incentivated the students throughout their education.

In regards to this paper, the authors would especially like to thank their supervisor Morgan Ericsson for always being available to help, from the conception of the research project to the submission of this paper. The authors would also like to thank K. M. Arif Aziz for always being ready to give feedback on the research in general both before and throughout the entire research project. Thank you as well to Kristoffer Chiem for always being there when it comes to supporting the paper until the very end.

Finally, the authors would also like to thank Justin Wagner and Benjamin No for the countless hours put into reviewing the paper towards the end of the project.

References

IEEE standard for a software quality metrics methodology. *IEEE Std 1061-1998*, 1998.

Introduction of quality requirement and evaluation based on ISO/IEC square series of standard. In *Trustworthy Computing and Services*, volume 320 of *Communications in Computer and Information Science*, pages 94–101. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-35794-7. URL http://dx.doi.org/10.1007/978-3-642-35795-4_12.

F. A. Administration. *Software Approval Guidelines*. U.S. Department of Transportation, 2003.

D. J. Barnes and T. Hopkins. Improving test coverage of lapack. *Applicable Algebra in Engineering, Communication and Computing*, 18(3):209–222, 2007.

W. C. Booth, G. G. Colomb, and J. M. Williams. *The Craft of Research*. Univ. of Chicago Press, 3 edition, 2003.

P. Bourque, A. Abran, J. Garbajosa, G. Keeni, and S. Beijun. Swebok: Guide to software engineering body of knowledge v3 (post-public review). URL https://computer.centraldesktop.com/home/viewfile?guid=42805308302A4B36ECC3C678DA51ED7DD384F9D8C&id=24071291, September, 2013.

P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *The Journal of Systems and Software*, 80:571–583, April 2007.

X. Cai and M. R. Lyu. The effect of code coverage on fault detection under different testing profiles. In *Proceedings of the 1st international workshop on Advances in model-based testing*, A-MOST '05, pages 1–7. ACM, 2005.

M.-H. Chen, M. Lyu, and W. Wong. An empirical study of the correlation between code coverage and reliability estimation. In *Proceedings of the 3rd International Software Metrics Symposium, 1996.*, pages 133–141, 1996.

M.-H. Chen, M. Lyu, and W. Wong. Incorporating code coverage in the reliability estimation for fault-tolerant software. In *Proceedings., The Sixteenth Symposium on Reliable Distributed Systems, 1997.*, pages 45–52, 1997.

F. Del Frate, P. Garg, A. Mathur, and A. Pasquini. On the correlation between code coverage and software reliability. In *Proceedings., Sixth International Symposium on Software Reliability Engineering, 1995.*, pages 124–132, 1995.

G. Fischer. The software technology of the 21st century: From software reuse to collaborative software design, 2001.

P. Garg. Investigating coverage-reliability relationship and sensitivity of reliability to errors in the operational profile. In *Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '94, pages 19–36. IBM Press, 1994.

M. Gittens, K. Romanufa, D. Godwin, and J. Racicot. All code coverage is not created equal: a case study in prioritized code coverage. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, CASCON '06. IBM Corp., 2006.

A. Gupta and P. Jalote. An approach for experimentally evaluating effectiveness and efficiency of coverage criteria for software testing. *International Journal on Software Tools for Technology Transfer*, 10(2):145–160, 2008. ISSN 1433-2779.

D. Hamlet. Connecting test coverage to software dependability. In *Proceedings., 5th International Symposium on Software Reliability Engineering, 1994.*, pages 158–165, 1994.

J. R. Horgan, S. London, and M. R. Lyu. Achieving software quality with testing coverage measures. *IEEE COMPUTER*, 27:60–69, 1994.

S. Inoue and S. Yamada. Testing-coverage dependent software reliability growth modeling. *International Journal of Reliability, Quality and Safety Engineering*, 11(4): 303–312, 2004.

M. Kessis, Y. Ledru, and G. Vandome. Experiences in coverage testing of a java middleware. In *Proceedings of the 5th international workshop on Software engineering and middleware*, SEM '05, pages 39–45. ACM, 2005.

Y. W. Kim. Efficient use of code coverage in large-scale software development. In *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '03, pages 145–155. IBM Press, 2003.

B. Kitchenham and S. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

B. A. Kitchenham. Procedures for performing systematic reviews. Technical Report TR/SE-0401, "Department of Computer Science, Keele University and National ICT, Australia Ltd, 2004.

S. Krishnamurthy and A. P. Mathur. On predicting reliability of modules using code coverage. In *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '96, pages 22–34. IBM Press, 1996.

J. Li. Prioritize code for testing to improve code coverage of complex software. In *16th IEEE International Symposium on Software Reliability Engineering, 2005.*, ISSRE 2005, pages 10 pp.–84, 2005.

M. Lyu. Software reliability engineering: A roadmap. In *Future of Software Engineering, 2007.*, FOSE '07, pages 153–170, 2007.

Y. Malaiya, N. Li, J. Bieman, R. Karcich, and B. Skibbe. The relationship between test coverage and reliability. In *Proceedings., 5th International Symposium on Software Reliability Engineering, 1994.*, pages 186–195, 1994.

Y. Malaiya, M. Li, J. Bieman, and R. Karcich. Software reliability growth with test coverage. *IEEE Transactions on Reliability*, 51(4):420–426, 2002. ISSN 0018-9529.

B. Marick. How to misuse code coverage. In *Proceedings of the 16th Interational Conference on Testing Computer Software*, pages 16–18, 1999.

A. Mockus, N. Nagappan, and T. T. Dinh-Trong. Test coverage and post-verification defects: A multiple case study. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, pages 291–301. IEEE Computer Society, 2009.

A. S. Namin and J. H. Andrews. The influence of size and coverage on test suite effectiveness. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, ISSTA '09, pages 57–68. ACM, 2009.

M. Newman. Software errors cost U.S. economy $59.5 billion annually - nist assesses technical needs of industry to improve software-testing. `http://www.abeacha.com/NIST_press_release_bugs_cost.htm`, 2002.

P. Piwowarski, M. Ohba, and J. Caruso. Coverage measurement experience during function test. In *Software Engineering, 1993. Proceedings., 15th International Conference on*, pages 287–301, 1993.

P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2): 131–164, 2009. URL `http://dx.doi.org/10.1007/s10664-008-9102-8`.

B. Smith and L. Williams. A survey on code coverage as a stopping criterion for unit testing. Technical Report TR-2008-22, Dept. of Computer Science, North Carolina State University, 2008.

A. Veevers and A. C. Marshall. A relationship between software coverage metrics and reliability. *Software Testing, Verification and Reliability*, 4(1):3–8, 1994.

L. J. White. Software testing and verification. *Advances in Computers*, 26:335–391, 1987.

T. Williams, M. Mercer, J. Mucha, and R. Kapur. Code coverage, what does it mean in terms of quality? In *Reliability and Maintainability Symposium, 2001. Proceedings. Annual*, pages 420–424, 2001.

A Templates

Data Item	**Value**	**Agreement (Y/N)**	**Additional Notes**
Name of paper			
Author(s)			
Journal			
Publication details			
Name of database(s)			
What data/results does the paper provide relevant to the correlation between code coverage practices and software quality?			
How does the paper define code coverage and its interpretation?			
What - if any - combinations of testing practices, including coverage practices, are suggested?			

Table 4: Template of Data Extraction Form