Human-based Test Design versus Automated Test Generation: A Literature Review and Meta-Analysis

Ted Kurmaku
Mälardalen University
Västerås, Sweden
tku21001@student.mdh.se

Eduard Paul Enoiu
Mälardalen University
Västerås, Sweden
eduard.paul.enoiu@mdh.se

Musa Kumrija
Mälardalen University
Västerås, Sweden
mka18003@student.mdh.se

ABSTRACT

Automated test generation has been proposed to allow test cases to be created with less effort. While much progress has been made, it remains a challenge to automatically generate strong as well as small test suites that are also relevant to engineers. However, how these automated test generation approaches compare to or complement manually written test cases is still an open research question. In the light of the potential benefits of automated test generation in practice, its long history, and the apparent lack of summative evidence supporting its use, the present study aims to systematically review the current body of peer-reviewed publications comparing automated test generation and manual test design performed by humans. We conducted a literature review and meta-analysis to collect data comparing manually written tests with automatically generated ones regarding test efficiency and effectiveness. The overall results of the literature review suggest that automated test generation outperforms manual testing in terms of testing time, the number of tests created and the code coverage achieved. Nevertheless, most of the studies report that manually written tests detect more faults (both injected and naturally occurring ones), are more readable, and detect more specific bugs than those created using automated test generation. Our results suggest that just a few studies report specific statistics (e.g., effect sizes) that can be used in a proper meta-analysis, and therefore, results are inconclusive when comparing automated test generation and manual testing due to the lack of sufficient statistical data and power. Nevertheless, our meta-analysis results suggest that manual and automated test generation are clearly outperforming random testing for all metrics considered.

ACM Reference Format:
Ted Kurmaku, Eduard Paul Enoiu, and Musa Kumrija. 2022. Human-based Test Design versus Automated Test Generation: A Literature Review and Meta-Analysis. In *ISEC 2022*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3511430.3511433

1 INTRODUCTION

Software plays a vital role in our daily lives and can be found in a number of domains, ranging from mobile applications to medical systems. In this context, software development organizations need to deliver reliable and high-quality software products while having to consider more stringent time constraints. This problem limits the amount of software testing that can be performed, and it needs to be managed through efficient and effective automated techniques. The creation of test cases in software testing is an intellectual activity in which engineers allocate a variety of cognitive resources when confronted with challenges as they go along. This is largely a manual activity, dependent on the ingenuity and thoroughness of humans. Automated test generation [2] has been proposed to allow test cases to be created with less effort.

The goal is to automatically find a small set of test cases that check the correctness of the system and guard against (previous as well as future) faults. While much progress has been made, it remains a challenge to create strong and small test suites that are also relevant to developers. Even so, mature tools for automated test generation are still few and consequently, the evidence regarding the comparison of automated test generation and manual testing is scarce. Thus, there is a need for a meta-analysis structuring this evidence and providing an overarching comparison.

In this paper, we present a meta-analytical approach on comparing two (and rather broad) classes of software test design techniques, specifically automated and manual test design, based on the findings of several primary studies. We conducted three separate meta-analyses with respect to mutation scores, decision coverage, and the number of tests produced by each testing technique. In addition, we used another statistical method, vote counting, and a qualitative synthesis method, narrative synthesis, as additional methods to obtain other reported results in our literature review. Our literature review results suggest that test cases generated by automated test generation tools are more cost-efficient in terms of coverage achieved, the number of tests generated, the time needed to create and execute tests and check the results. In addition, our results indicate that test cases created manually by humans achieve similar or better fault detection scores when compared using both injected or naturally occurring faults. Even though these fault detection scores are higher for manual test design on average, there is no statistically significant difference between these two techniques. Given that most of the studies do no report statistics to be used in a meta-analysis, our results are limited. There is a need for more empirical studies that are properly documented in order to perform a full-blown meta-analysis on the comparison between manual and automated test generation.

2 PLANNING THE LITERATURE REVIEW AND META-ANALYSIS

The method used in this paper is based on the guidelines for a literature review and meta-analyses proposed by Barbara Kitchenham

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9618-9/22/02.
https://doi.org/10.1145/3511430.3511433

[23]. Meta-analysis refers to the process of finding, selecting, analyzing, and combining information relevant to a particular research question or goal. The meta-analysis begins with formulating a problem, search the literature, selecting the studies, extracting values such as differences and means, and the statistical analysis and calculations [23]. Then, the combination of p-values from independent tests over the same hypotheses regarding cost and fault detection is performed. Since combining p-values may not always be helpful when the effects of the difference between manual and automated test generation in the combined studies are not consistent, the effect sizes of different studies can be combined to have a clearer overview of the overall effect size of these differences. Two approaches can be used when pooling the effect sizes for meta-analysis: The Fixed Effect Model or the Random Effect Model [5]. The fixed-effects model presumes that all studies and their effect sizes derive from a homogeneous population. However, in the random-effect model, the study effect has a more significant variance than the data drawn from the same population. In fact, in the random-effects model, that data is drawn from a vastly heterogeneous population.

2.1 Research question

This paper addresses the following research question: *RQ: How does automated test generation compare with manual testing in terms of cost and fault detection?* This question is essential since the emerging evidence comparing automated test generation and manual testing is scarce. Thus, there is a need for a literature review and a meta-analysis structuring this evidence and providing an overarching comparison.

2.2 Search process

We used the PICO (Population, Intervention, Comparison, and Outcomes) [24] approach to identify keywords from the research question in order to create the search string as follows: Population (i.e., engineers using automated and manual test design approaches), Intervention (i.e., the automated and manual test design approaches and tools), Comparison (i.e., experiments on cost and efficiency), Outcomes (e.g., metrics related to cost and efficiency). Thus, the identified keywords are automated, manual, generate, test, software, and compare.

We used the following search string, which gave the best results for covering a predefined set of manually searched papers on this topic:

test AND generation AND (manual OR handcrafted OR handmade OR manually) AND software AND (evaluation OR experiment OR case study OR comparative OR comparison)

We selected IEEE Xplore and ACM Digital Library as the digital libraries and Scopus as the indexing system. The databases have been selected based on the experience reported by Dyba et al. [11] as well the authors' knowledge about the previous studies published on this subject. The search process was performed in March 2020. The search results per each database are reflected in Table 1.

Table 1: Number of studies per database.

Database Name	Results
IEEE Xplore	226
ACM Digital Library	344
Scopus	217

2.3 Study selection

The study selection process starts with removing the studies that are irrelevant from the set of the identified candidate papers. After the first screening, the papers need to be looked at in more detail. The process continues with reading the titles and abstract, going deeper into the introduction, and full-text reading. For our set of papers, we took the following steps: First, the duplicates were removed with the help of the software. Second, the titles were read to determine whether the articles focused on testing, automated or manual testing, and test generation in the scope of software engineering. Third, we read the abstracts to check whether the studies had an empirical background and were not secondary studies, whether they contained the search words, and made a comparison of manual and automated tools or techniques on software or systems. It is important to note that we found several papers that were not easy to understand whether they were relevant to our study by reading the abstract. We performed a kappa analysis to check our level of agreement. To complete this procedure, we achieved a kappa value k=0.608, which indicates a substantial strength of agreement. In addition, we found several papers that reported more than one study. As suggested by Kitchenham [24], these papers can be considered as separate studies for the sake of the analysis. This fact was used in the meta-analysis process, as reported in the later sections of this study. Another thing worth mentioning is that we also came across several papers, which were replications of a previous study, or reporting an experiment conducted in a certain period, and later this experiment was replicated. Thus, both experiments were reported in a later study. We decided to keep only the latest version for such papers since these papers can be considered replications of previous studies.

The following inclusion criteria were applied to each paper during the selection process: studies that compare manual testing and automated testing, model-based testing and manual testing, random testing and manual testing, and studies that present the result of an empirical study in software engineering published after 2010[1]. The exclusion criteria include the following categories: studies presenting summaries of conferences/editorials, non-peer-reviewed material, studies not accessible in full text, books, and gray literature, studies that are duplicates of other studies, and secondary studies.

We extracted the following data from each study: the source of the paper (journal or conference), the method used to conduct the study (experiment, case study, survey), the type of comparison (manual and automated, manual and model-based testing), the program or system under test, the language used for programming of each system under test, the size of each program (LOC, classes,

[1]We included publications after 2010 since we wanted to focus on recent and relevant comparison studies comparing the latest automated test generation approaches with manual testing.

methods). Many studies suggested this process has two actors, the extractor and the reviewer [23] [24], but we preferred to do both roles for each process. We first extracted all the needed information for the paper and checked the accuracy of the results. When there was a disagreement, one's reasoning and interpretation were discussed until the first and the last authors agreed.

2.4 Meta Analysis

Meta-analyses use statistical methods to analyze and pool the results from several primary studies, usually experiments, that have measured the outcome of two different treatments or interventions. Meta-analyses provide a higher statistical meaning for their outcomes than individual primary studies [24]. Unfortunately, not too many meta-analyses are reported in software engineering. Since there is no clear understanding of what a representative sample should look like [20], the results of the primary studies are incomparable, and providing relevant knowledge through these methods is quite challenging.

2.5 Study Selection Process Results

Figure 1 shows a flow diagram representation of the study selection process and the number of papers that resulted in each step. The initial set of papers consisted of a total number of 787 papers. We removed duplicates and ended up having a total of 682 papers. In the next step, we applied the inclusion and exclusion criteria. First, we remove non-peer-reviewed papers and those published before 2010 (this resulted in 470 papers). Second, we checked if the tags and keywords contained any of the keywords identified in the search string (first searching for the keywords "software", "programs", and "systems", which resulted in 236 remaining papers, and second using the "test" or "testing" keywords, which resulted in 193 results). Third, the title and the abstract were examined to remove papers using the exclusion criteria (this step resulted in 155 papers). We checked the papers' full text in the next step and arrived at our final set of 29 papers. Quality assessment was the last step and the set of papers was reduced to 17 papers. The search process had not come to an end yet. The first steps of the process were done separately; however, we had a few papers that we found necessary to examine in pairs. These were the papers that were not clear, and we named this process the "excluded-papers review". These papers were not excluded right away; instead, they were kept apart for further examination. The full text of these papers was read in pairs, followed by a discussion of whether the papers were relevant to our study or not. This process concluded with four more papers to add to the set. Therefore, the final set consisted of a total of 21 papers.

3 LITERATURE REVIEW RESULTS

This section presents the results from the analysis of the 21 primary papers included during the study selection process (Table 2).

3.1 Description of the Primary Studies

P1: [12]: In this paper, the authors investigate how the automated test cases compared to the manual tests in terms of cost and effectiveness. The main area of their investigation is the control software used in the industry. They test 61 real-world industrial programs written in IEC 61131-3. The automatically generated test cases reach a similar code coverage with the manually written tests, but they generate the test cases almost 90% faster.

P2: [13]: In this paper, the authors investigate specification and implementation-based testing on two programs of safety-critical software systems, written in IEC 61131-3 language. Their goal is to measure the efficiency and effectiveness of fault detection by conducting a controlled experiment using a test written by twenty-three master students of software engineering.

P3: [37]: In this paper, the authors describe the comparison of the tool-supported test case and manually written unit tests. This paper is a replication of an empirical study that describes an experiment involving professional engineers with several years of experience in the software industry compared to the students from the initial study and an extended time limit. The paper investigates the differences between students from the initial study and professionals from this study and the extended time limit impact.

P4: [29]: In this paper, the authors present a language framework that creates mappings for concrete tests automatically based on abstract tests. They address three issues: creating mappings and test value generation, transforming graphs, and using coverage criteria to generate test paths and solving constraints, and generating real tests.

P5: [17]: In this paper, the authors compare the readability of manually written test cases and the classes they test and further, examining the readability of automatically generated test cases. An exploratory study is conducted to obtain the results. The results suggest that developers often neglect readability and that the automatic test cases are much less readable in general than those written manually.

P6: [16]: In this paper, the authors propose a plan and design an experiment to analyze a test case generation strategy in order to evaluate its completeness from the point of view of the tests, which will use the Communication Analysis-based requirements model. The testers apply two different methods for obtaining the test cases manually; the first is without derivation rules, and the second is using transformation rules. After obtaining the manually derived test case, a comparison against the test cases of automated generation using transformation rules is made.

P7: [39]: In this paper, the authors compare manually written test cases with model-based testing on two different versions of a web-based data collection and review system. The experiment was conducted with developers who tested each software version, having identical testing goals and resources, but the technique used was different. The first developer used manual testing, and the second used model-based testing with automated test generation tools. The authors compare the effectiveness of defects found and efficiency in terms of effort spent to complete each process.

P8 [9]: In this paper, the authors propose an automated model-based approach to generate test inputs for testing new data requirements by modifying the old data.

P9: [30]: In this paper, the authors design and conduct an empirical experiment to compare manual ad hoc testing with model-based testing with the aid of the TaRGeT tool. The metrics measured by these experiments are the time spent by testers and the number of found defects. These metrics were analyzed for their correlation with testers' experience and the complexity of the system under test.

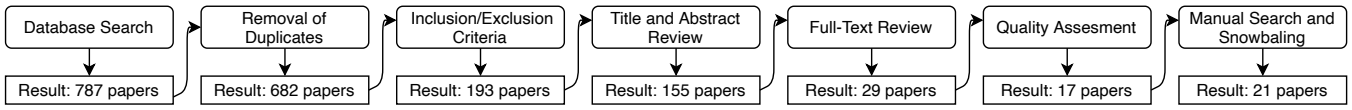

Figure 1: Number of studies during the study selection process.

The system under test is a real-life project with 54 experimental units, 27 use cases, and 82 new defects detected.

P10: [6]: In this paper, the authors conduct three experiments with five replications with a total of 55 human subjects to evaluate the impact that properties of automated tests such as readability and understandability, have on the effectiveness and efficiency of debugging. They conducted two experiments using automation tools such as Randoop and EvoSuite, while the third experiment explores the code identifiers role in test cases since this is the main difference between the two types of test cases.

P11: [14]: In this paper, the authors perform two controlled experiments with 97 human subjects to compare the manually written test cases with test cases generated automatically with the aid of tools in terms of coverage, number of bugs, number of tests, effectiveness, and mutation.

P12: [26]: In this paper, the authors compare test suites generated from automated test generation tools with test suites written manually by the developers in terms of complexity and quality. This empirical study examines ten programs with existing test suites and creates automatically generated test suites. The test suites are measured in terms of code coverage and fault detection. The tool used for this type of testing are CodePro and EVOSUITE.

P13: [45]: This study compares a Dynamic Symbolic Execution (DSE) tool, KLEE, which is an automated test generation tool with the manually generated test suites on several GNU CoreUtils programs. This paper covers different aspects such as efficiency, coverage, size of the test, and readability.

P14: [10]: This paper compares tests generated from a model-based testing tool, FormTester, with the manually generated tests. The study is conducted testing several online applications, both online and offline. While manual testing is performed under two projects, small and one medium to large size, the FormTester is considering only one project with a bigger number of LOC.

P15: [42]: This paper reports the results of a survey that compares different android testing techniques. The survey's focus is to highlight the developers' preferences for the design and test generation, manually and automatically, and their perceptions of different metrics. In mobile apps, the testing is usually performed manually, which makes it a costly activity.

P16: [46]: In this paper, the authors compare a model-based testing tool, TesMa, with manual testing. They compare the approaches in terms of effectiveness and cost. They are faced with two challenges. The first one is that describing the design model in the unfamiliar modeling tool or complicated notations will be difficult, and second, for any specific domain, multiple viewpoints of test cases should be considered. The system under testing is a traditional one, developed manually.

P17: [41]: This paper is a replication of a previous study conducted ten years ago in which the authors compared automated test generation tools with the manually written test suites. The authors used the same tools and experimental environment as the former authors did, but with the updated version, including some new tools released in the last years.

P18: [35]: In this paper, the authors evaluate several test suites in terms of error detection, model, and implementation coverage. This paper makes a comparison of automatic and manual testing, but it goes further, comparing these suites when they use models and when they do not use models.

P19: [36]: In this paper, the authors conduct an experiment between 48 masters students that have 60 minutes to write manual unit tests and an automated test generation tool called Randoop, which generates an amount of tests in only 2 minutes. After the test is generated, they run the tests to see which techniques are more effective, have greater coverage and fault detection.

P20: [28]: In this paper, the authors talk about a model-based testing tool, Skyfire, which can automatically generate effective Cucumber test scenarios to replace manually generated test scenarios. The system under test is Roc, written in Java, and serves as an infrastructure for big data applications. The developers write Cucumber mappings from tests generated by Skyfire, and they present the implementation and design of the former.

P21: [27]: In this paper, the authors focus on Web applications and their Web pages and view a test scenario as a sequence of steps that are taken to transition from one Web page to another. They compare a tool they have created with the manually generated test from several engineers and evaluate their approach by comparing it with manual generation and two open-source tools and making a qualitative evaluation by experienced developers.

3.2 Experimental Characteristics and Reported Measurements

The final set of 21 papers includes 17 conference papers and four journal papers. Regarding the empirical methods used to obtain the results, we found 15 papers that conducted experiments, eight papers used a case study, and only one performed a survey. In addition, 13 papers compared code-based automated test generation with manual testing, whereas eight papers made this comparison with model-based testing. Most of the human subjects were students, followed by researchers, developers and testers.

The programs under test are mainly open source in the majority of the papers, with just a few studies focusing on proprietary and commercial systems. Most papers use more than one program or class to run their tests on. The main targeted programming language is Java, with only a few papers targeting testing for different programming languages in industrial environments. The size of

Table 2: The final set of studies included during the study selection process

Title	Type	Method	Type of comparison
P1: A Comparative Study of Manual and Automated Testing for Industrial Control Software	Conference paper	Case study	automated test generation and manual testing
P2: A Controlled Experiment in Testing of Safety-Critical Embedded Software	Conference paper	Experiment	automated test generation and manual testing
P3: A Replicated Study on Random Test Case Generation and Manual Unit Testing: How many bugs do professional developers find?	Conference paper	Experiment	automated test generation and manual testing
P4: A Test Automation Language Framework for Behavioral Models	Conference paper	Experiment	automated test generation and manual testing
P5: An Empirical Investigation on the Readability of Manual and Generated Test Cases	Conference paper	Experiment	automated test generation and manual testing
P6: An Experiment Design for Validating a Test Case Generation Strategy from Requirements Models	Conference paper	Experiment	model-based testing and manual testing
P7: Assessing model-based testing: An empirical study conducted in industry	Conference paper	Case study	model-based testing and manual testing
P8: Augmenting Field Data for Testing Systems Subject to Incremental Requirements Changes	Journal article	Industrial Case study Experiment	model-based testing and manual testing
P9: Comparing Model-Based Testing with Traditional Testing Strategies: An Empirical Study	Conference paper	Experiment	model-based testing and manual testing
P10: Do Automatically Generated Test Cases Make Debugging Easier? An Experimental Assessment of Debugging Effectiveness and Efficiency	Journal article	Experiment	automated test generation and manual testing
P11: Does Automated Unit Test Generation Really Help Software Testers?	Journal article	Experiment	automated test generation and manual testing
P12: Empirically Evaluating the Quality of Automatically Generated and Manually Written Test Suites	Conference paper	Experiment	automated test generation and manual testing
P13: Experience Report: How is Dynamic Symbolic Execution Different from Manual Testing? A Study on KLEE	Conference paper	Experiment	automated test generation and manual testing
P14: FormTester: Effective Integration of Model-Based and Manually Specified Test Cases	Conference paper	Experiment	model-based testing and manual testing
P15: How do Developers Test Android Applications?	Conference paper	Survey	automated test generation and manual testing
P16: Introducing Test Case Derivation Techniques into Traditional Software Development Obstacles and Potentialities	Conference paper	Case study	model-based testing and manual testing
P17: On the Effectiveness of Manual and Automatic Unit Test Generation: Ten Years Later	Conference paper	Case study Experiment	automated test generation and manual testing
P18: One Evaluation of Model-Based Testing and its Automation	Conference paper	Case study	model-based testing and manual testing
P19: Random Test Case Generation and Manual Unit Testing: Substitute or Complement in Retrofitting Tests for Legacy Code?	Conference paper	Experiment	automated test generation and manual testing
P20: Skyfire: Model-Based Testing with Cucumber	Conference paper	Industrial Case study	model-based testing and manual testing
P21: Test scenario generation for web application based on past test artifacts	Journal article	Case Study Experiment	automated test generation and manual testing

the programs ranges from hundreds to tens of thousands of lines of code.

In most cases, the type of defects used for fault detection is seeded defects, with just a few studies focusing on naturally occurring faults.

In addition, just a minority of papers included report statistics such as effect sizes and p-values. These papers produce a more specific set of results regarding different metrics such as mutation score, coverage, and the number of tests. Four of the papers which report statistical data were used to perform the meta-analysis. Other papers that do not report effect sizes but report other statistics can be used to draw several more specific conclusions. We evaluated papers in the final set that provided valuable data and information to answer our research question. They report empirical methods of high quality with valuable results.

3.3 Literature Review Results

The overall results show that most of the studies do not report a significant difference between manual and automated test generation for all metrics considered (studies do not show any statistical significant difference when looking at p-values for fault detection, code coverage, and cost metrics[2]. On the other hand, when comparing p-values for specific metrics, the results suggest that automated test generation outperforms manual test design in terms of the number of tests, testing time and coverage achieved. One metric that shows that manual testing is outperforming automated test generation is related to the readability of the resulting test cases.

> *Regarding fault detection effectiveness (for both injected faults and naturally-occurring faults), 14 out of 21 papers report that manual tests outperform automated generated tests. Automated test generation outperforms manual test design in only two studies.*

Although some of these results are significant, these calculations typically rely on different contexts, tools, SUTs, and statistical model assumptions. Drawing reliable conclusions from these results is a difficult task, but we are indicating that the research in automated test generation and software testing must improve its empirical reporting practices to improve these much-needed literature review studies. In the next section, we show the results of the meta-analysis process and the challenges faced during this endeavour.

4 META-ANALYSIS RESULTS

We performed the meta-analysis on four out of the twenty-four studies identified as our primary studies: P1 [12], P2 [13], P10 [6] and P11 [14]. A crucial aspect of each meta-analysis is that all the primary studies must use the same effect size. Keeping this in mind, each paper was carefully examined, and we selected only the papers which reported statistical values suitable for our analysis and more importantly, the same effect size. We extracted the p-values and the effect sizes of each paper. We noticed that two of the four papers reported two replications of experiments, so we considered the values of each experiment separately.

The studies were not dependent on previous studies; therefore, we excluded the possibility of having a multi-level meta-analysis. We found two types of data reported in our set of primary studies. The first type of data was "pre-calculated effect size data" and the second was "raw effect size" data. The pre-calculated effect size data is straightforward, and we only reported it on the table of the effect sizes. Raw effect size data, on the other hand, comes in different formats. First, the "standard effect size data" reports values such as mean, standard deviation, and sample size. The second type is event rate data, which reports the number of events on the control and intervention groups and the number of groups in each trial. The third type of raw effect size data is the incidence rate data. This set of data includes values such as the number of events and the person-time at risk. It was easily understandable that incidence rate data was the kind of data that we could find in our studies since the set of our primary studies is related to software engineering. We examined the papers with raw effect size data and identified that the data in these papers belonged to the standard effect size data thus, mean, standard error, and the sample size of each technique was extracted.

The effect size is often referred to as Cohen's d, which was the case in our primary studies, which reported pre-calculated effect sizes (i.e., Cohen's d estimator). Because all the primary studies must use the same effect size, and the other studies reported effect sizes based on Cohen's d, we calculated the effect sizes for the studies that only reported raw effect size data.

We used the mean of each group, control and treatment, the standard deviation of each group, and the sample size. This calculation was made for all four papers, and the values were reported in the table. The core of every meta-analysis is pooling the effect sizes in order to get one overall effect size estimate of the studies [18]. When pooling the effect sizes, we can use two approaches, the Fixed-Effect Model or the Random-Effects Model [5]. The calculations used for the meta-analysis do not rely on the effect size metric used; they rely on the model used [24]. There is a considerable debate on which one of the approaches should be used. The fixed-effects model assumes that all studies, along with their effect sizes, derive from a single homogeneous population [5]. In practice, this is not realistic since the sample size or its methods can vary, and in this case, we cannot consider that the sample comes from a homogeneous population. In our study, we identified as sample size the various programs and systems under test. Since the programs under test in our studies are different in terms of programming languages, even though we did not have a significant number of effect sizes to pool, we decided to proceed with the random effect size approach. We want to explain and defend the assumption that variance is higher than when the results are derived from a single population [40]. The random effect size model demonstrates that there is another reason for the variance to be present besides the sampling error, and this is the fact that studies do not derive from a single population; they are gathered from an enormous population [18]. However, there is a drawback of using this method because this method gives more attention to small studies, which are considered more bias-prone than the larger studies. In order to have a pool of effect sizes to get one overall effect size estimate of the studies, besides the effect size values from each study, it is necessary to have the Standard Error of each study as well. The standard error uses the standard deviation to measure the accuracy of a sample distribution representing a population. It depends both on the sample standard deviation as well as on the sample size. The standard error value decreases when the sample size increases, but this event does not affect the effect size value. The studies we selected for the meta-analysis did not directly report the Standard error; they reported only the effect size and the p-value. The studies

[2]We use a traditional statistical significance limit of 0,05

reported effect size values for the following metrics: Fault detection (in terms of number of bugs detected and mutation score), number of tests, debugging effectiveness and efficiency, and coverage – code, statement, branch, and method. In the end, the only metrics we could perform a meta-analysis for all papers are decision coverage, mutation score, and the number of test cases.

4.1 A Meta-Analysis on Branch (Decision) Coverage

We performed the first meta-analysis on three studies [12] [13] [14]. First, we pooled the effect sizes to get an overall effect size estimate. The "meta" library was used to complete the task. The analysis was easy to code in R, but there were three parameters we had to take care of. The first is the "between-study-variance estimator", $\tau 2$. This is the effect size variance across the study population and reflects the variance of the true effect sizes. Several estimators are available, but as other studies suggested [18], we chose the Sidik-Jonkman estimator ("SJ") and the HKSJ method for the necessary adjustments.

Our meta-analysis's second step is to visually present the data using forest plots (as shown in Figure 2). There are several statistics present in the forest plot. Besides the study label, effect size TE, and standard error SE, we show several different values such as the weight of each study, the standard mean difference, and the 95 confidence interval. The red boxes show the individual effect size of each study, while the black diamond shows the pooled effect size. The bigger the box means the study weighted more because the sample size was more significant. We can see the more considerable weight in the first study since the sample size was more significant. The first study had a sample size of 61 programs. The horizontal line across the box of each study illustrates the length of the confidence interval. In this case, the longer the line across the box of each study, the less accurate the study findings are. Again, we can see that the findings of the first study are accurate. Consequently, the horizontal line is minimal. The same applies to the width of the diamond. The wider the diamond, the less accurate the meta-analysis is. We can see from the plot, the width of the diamond is small. Therefore the results of this meta-analysis are reliable. The vertical line is the line of no effect, which means this is the position where there is no apparent difference between the two techniques. If the outcome of interest is unfavorable, the results on the left encourage the intervention over the control group meaning the outcome of interest, which in our case is the coverage, is lower in the intervention group compared to the control group. In our case, we consider random testing as treatment and automated and manual testing as the control group. If the outcome of interest is advantageous, the results on the right encourage the treatment over the control group, meaning the outcome of interest appeared more often in the intervention group than the control group. In our forest plot, we notice that all the studies appear on the right side of the vertical line. These results suggest that the manual or automated testing techniques showed better decision coverage than the random testing. The pooled effect size sits at 0.2 with a 95% confidence interval between -0.22 to 0.62. The effect size is small, meaning that the difference is trivial.

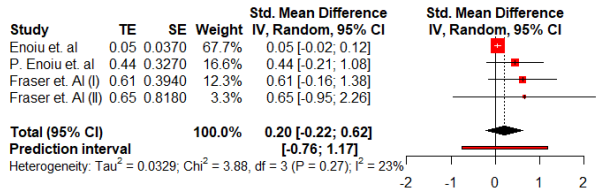

Figure 2: The graphical representation of the pooled effect size, known as forest plot, along with several statistical data.

4.1.1 Heterogeneity. In a meta-analysis, heterogeneity refers to the variation in study outcomes between studies. Between studies, heterogeneity is one of the biggest concerns when doing a meta-analysis. Meta-analysis has often been criticized for combining studies that are very different. High heterogeneity within studies means that the studies have nothing in common, and it is meaningless to pool their effect size. Therefore, we need to check that the effects sizes of each study are similar, and if the variance is present, it happens only due to randomness. In other cases, studies report extreme effect sizes, which were included in the meta-analysis. These values are called outliers, which can distort the overall effect size of the study. Three types of heterogeneity measures assess the scale of heterogeneity. The most common and classical method to measure heterogeneity is Cochran's Q. Higgin's & Thompson's I2 is another method for calculating the heterogeneity between studies. It is the percentage of variability of the effect sizes which is not caused by sampling error. The third measure is tau squared T2. We see the output of pooling effect sizes already gives us all three measures of heterogeneity: T^2, as we can see from tau^2, has a value of 0.0329, I^2 is reported under I^2 and the value reported is 22.6% with a confidence interval between 0.0% and 88.2% and Cochran's Q which is reported as Q under the test of heterogeneity has a value of 3.88 and a p-value equal to 0.275.

The result suggested that there were no outliers among our studies. Therefore we can confirm once again that the results are robust. However, as we mentioned before, outliers are not the only reason that may compromise the robustness of the pooled effect size outcome. Influence analysis is an analysis that checks if the pooled effect size relies heavily on a single study.

Figure 3 presents how the results would have changed if one of the studies were omitted. We can see that if excluded from our meta-analysis the Enoiu et al.[12] study, which is the study with the most significant weight, the pooled effect size would have changed from 0.20 to 0.52. Thus, we can confirm that this is the study with the most significant influence on the analysis.

4.2 A Meta-Analysis on Mutation Analysis

A meta-analysis on the mutation score was performed on two studies [12] [14]. Two studies are not enough to perform a meta-analysis, but the second study is considered two different studies since the experiment is conducted twice. Again, we have named it Fraser et al. [14] (I) for statistical values deriving from the first experiment, and Fraser et al. [14] (II) for statistical values deriving from the second experiment. For the experiments that do not report overall statistics, we took the average statistics reported for the tested

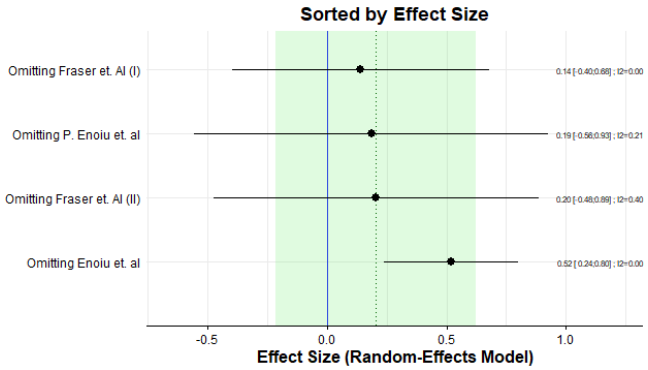

Figure 3: A graphical representation of influence analysis for decision coverage.

systems. For the first study, the programing language is IEC61131-3, and for the second study, which reported two experiments, it is Java. For the "between-study-variance estimator" $\tau 2$, we chose the Sidik-Jonkman estimator ("SJ") and the HKSJ method for the necessary adjustments. The mean differences were chosen as the summary measure, and the R code was executed. We visualize the results on a forest plot for a clearer presentation in Figure 4.

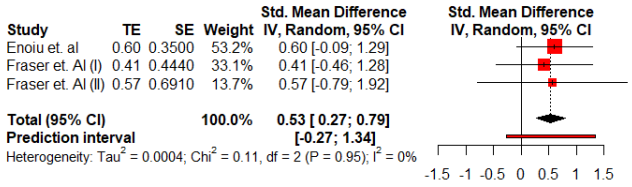

Figure 4: The graphical representation of the pooled effect size, known as a forest plot, along with several statistical data for mutation score.

The red boxes, which represent the weight of each study, show that the first study Enoiu et al. [12] had the most significant weight since the sample size is larger than the other studies, with a number of 61 subject programs. The horizontal line suggests the most accurate study is the first one since the line is shorter. The black diamond representing the pooled effect size of all studies has a small width, suggesting that the meta-analysis results can be considered reliable. The vertical striped line represents the line of no effect. Like the first meta-analysis, we consider random testing as treatment and automated and manual testing as the control group. The forest plot shows that all the studies are on the right side of the vertical line, meaning that manual or automated testing shows a higher mutation score than random testing. The pooled effect size shows a value of 0.53, with a 95% confidence interval from 0.27 to 0.79. This is a medium-size value, meaning that the difference is noticeable.

4.2.1 Heterogeneity. The next step is to calculate the between-study heterogeneity. R Studio calculates the heterogeneity by default when we pool the effect sizes. The outcome reports the values of all three measures of heterogeneity: T^2 has a value of 0.0004, I^2 reported a value equal to 0.0% with a confidence interval between 0.0% and 88.2% and Cochran's Q has a value of 0.11 and a p-value equal to 0.945. These values suggest that we have a low to no heterogeneity among our studies. The results are robust, which can also be used in future scenarios.

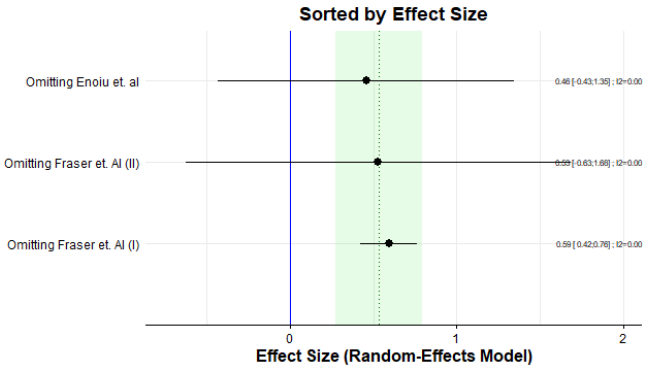

Figure 5: A graphical representation of influence analysis for mutation score.

Figure 5 suggests the results of the first study has the strongest influence on the effect size on the pooled effect size, yet, examining the chart from the table, we could confirm that omitting any of the studies, the pooled effect size remains almost in the same interval.

4.3 A meta-analysis on the number of tests

Meta-analysis using the number of tests was performed on three studies [12] [13] [14]. The third study is considered two different studies since the experiment is conducted once and replicated with other participants. We have named it Fraser et al. [14] (I) for statistical values deriving from the first experiment, and Fraser et al. [14] (II) for statistical values deriving from the second experiment. The average of the statistics reported for the systems under test was calculated for the experiments that do not report overall statistics. We visualize the results on a forest plot for a clearer presentation in Figure 6.

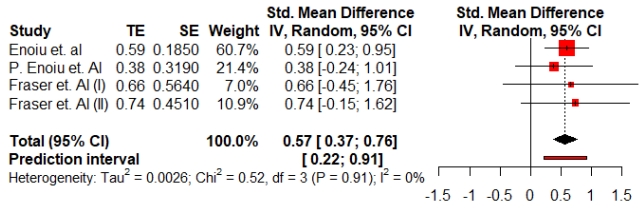

Figure 6: The graphical representation of the pooled effect size, known as forest plot along with several statistical data for the number of tests.

The red boxes, which represent the weight of each study, show that the first study Enoiu et al. [12] had the most significant weight since the sample size is larger than the other studies, and the number

of subjects was 61. The horizontal line suggests the most accurate study is the first since the line is shorter. The black diamond representing the pooled effect size of all studies has a small width, suggesting that the meta-analysis results can be considered reliable. The vertical striped line represents the line of no effect. We consider random testing as treatment and automated and manual testing as the control group like the first two meta-analyses. The forest plot shows that all the studies are on the right side of the vertical line, meaning that manual or automated testing shows a larger number of tests than random testing. The pooled effect size shows a value of 0.57, with a 95 confidence interval from 0.37 to 0.76. This is a medium-size value, meaning that the difference is noticeable.

4.3.1 Heterogeneity. R Studio calculates the heterogeneity by default when plotting the effect sizes. The outcome reports the values of all three measures of heterogeneity: T^2 has a value of 0.0026, I^2 reported a value equal to 0.0 with a confidence interval between 0.0 and 11.7 and Cochran's Q has a value of 0.52 and a p-value equal to 0.914.

The extreme-size cases and outliers' analysis among the studies did not report any value. We did the influence analysis, which shows the study which has the most significant impact on the overall effect size and presented it graphically in Figure 7.

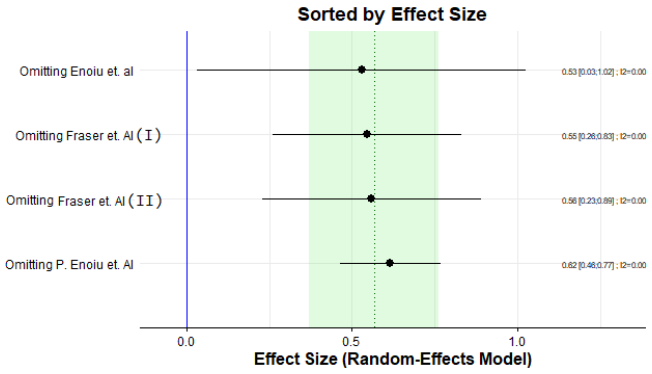

Figure 7: A graphical representation of influence analysis for mutation score.

The second study has the biggest influence on the overall effect size. Omitting this study, the overall effect size value increases from 0.57 to 0.62, with a 95 confidence interval of 0.46 to 0.77.

5 DISCUSSION

In this section, we present an outline of the collected evidence regarding our analysis. This information was collected when performing the literature review (i.e., short summary an collected statistics) and meta-analysis.

Based on the literature review results, it seems that manual tests achieve higher mutation and fault detection scores when compared with automated test generation. The vast majority of the papers report that automated tests are slightly worse in fault detection than manual testing, while manual tests are highly effective in detecting specific types of faults. The test suite size is reported to impact fault detection; however, when test suites with equal size were used, the fault detection achieved by automated tests was still not showing improvements compared to manual tests. When the code is hard to cover and the mutants are hard to kill, using manual tests shows better results, as reported in several studies. Even though manual tests achieve higher fault detection, the automatically generated tests expose complex scenarios. In addition, manually written tests obtain better fault detection scores even compared to pure random testing. However, when manual and automated tests can detect similar faults, the preferred technique would be the latter since the tests are generated faster.

We observed benefits in terms of cost savings in terms of testing time. For manual testing, all costs are related to human effort, whereas automated test generation costs involve both machine and human resources. Based on the results reported, the creation, execution, and reporting costs are very low when using automatic test generation tools compared to manual testing. On the other hand, the relative difficulty of understanding each created test increases the cost of checking the results for automated test generation. Thus, even though automated tests often result in more code lines than manual tests, the test generation tools can produce high-quality test suites with reduced costs and human effort.

> *In our meta-analysis, results suggest that manual and code-based automated test generation techniques are better than random testing in terms of injected faults detected (using mutation) and coverage achieved.*

Overall, the validity of the included studies and the limited number of studies included in the meta-review threaten the validity of our results. Even if the results of the meta-analysis are limited, the primary purpose of conducting this study was to review the literature and identify the main requirements for conducting consistent experiments in comparing manual test design and automated test generation.

> *We recommend researchers performing empirical studies on automated test generation to include statistical values suitable for a meta-analysis when comparing their technique with manual test creation (e.g., the p-values and the effect sizes)*

5.1 Limitations

There are two types of limitations present in our study: the primary studies and the review levels. The primary studies' quality, the variety of the research questions these studies are posing, the empirical methods used, and the study participants are some of the main concerns regarding the limitations of the primary studies. The participants chosen for the empirical methods were primarily students. This factor could directly impact the outcome of these primary studies. In addition, only a few of them report statistics that are much needed to perform a meta-analysis. In our case, we found twenty-one primary studies with only seven reporting statistics, and from this set of seven papers, we could combine statistical data for our meta-analysis for only four of them. Also, the setup of these studies was different in terms of the conditions and the tools being used. Regarding the limitations at the review level, when performing the search, deciding about inclusion and exclusion, and making various decisions during synthesis, the results could be

influenced by bias. We tried to minimize these as much as possible, with several reviews and discussions for each paper.

6 CONCLUSIONS AND FUTURE WORK

Our literature review results suggest that automated test generation outperforms manual testing in terms of cost, code coverage, and the number of tests created. On the other hand, in most of the studies, manually written tests achieve better fault detection on average. When using injected faults, manually written tests seem to also achieve a higher mutation score than automatically generated tests. Our results also show that few studies report the needed statistics to perform a meta-analysis. More studies are needed to structure the evidence in this area.

ACKNOWLEDGMENTS

The work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 957212, by Software Center and Vinnova (XIVT and SmartDelta).

REFERENCES

[1] [n. d.]. Decision Coverage Testing. https://www.tutorialspoint.com/software_testing_dictionary/decision_coverage_testing.htm
[2] Saswat Anand, Edmund K Burke, Tsong Yueh Chen, John Clark, Myra B Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil McMinn, Antonia Bertolino, et al. 2013. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software* 86, 8 (2013), 1978–2001.
[3] Luciano Baresi and Mauro Pezze. 2006. An introduction to software testing. *Electronic Notes in Theoretical Computer Science* 148, 1 (2006), 89–111.
[4] Angela Boland, Gemma Cherry, and Rumona Dickson. 2017. Doing a systematic review: A student's guide. (2017).
[5] Michael Borenstein, Larry V Hedges, Julian PT Higgins, and Hannah R Rothstein. 2011. *Introduction to meta-analysis*. John Wiley & Sons.
[6] Mariano Ceccato, Alessandro Marchetto, Leonardo Mariani, Cu D Nguyen, and Paolo Tonella. 2015. Do automatically generated test cases make debugging easier? an experimental assessment of debugging effectiveness and efficiency. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 1 (2015), 1–38.
[7] James M Clarke. 1998. Automated test generation from a behavioral model. In *Proceedings of Pacific Northwest Software Quality Conference. IEEE Press*. Citeseer.
[8] Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. 1978. Hints on test data selection: Help for the practicing programmer. *Computer* 11, 4 (1978), 34–41.
[9] Daniel Di Nardo, Fabrizio Pastore, and Lionel Briand. 2017. Augmenting field data for testing systems subject to incremental requirements changes. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 1 (2017), 1–40.
[10] Rahul Dixit, Christof Lutteroth, and Gerald Weber. 2015. FormTester: effective integration of model-based and manually specified test cases. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 745–748.
[11] Tore Dyba, Torgeir Dingsoyr, and Geir K Hanssen. 2007. Applying systematic reviews to diverse study types: An experience report. In *First international symposium on empirical software engineering and measurement (ESEM 2007)*. IEEE, 225–234.
[12] Eduard Enoiu, Daniel Sundmark, Adnan Čaušević, and Paul Pettersson. 2017. A comparative study of manual and automated testing for industrial control software. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 412–417.
[13] Eduard P Enoiu, Adnan Cauevic, Daniel Sundmark, and Paul Pettersson. 2016. A controlled experiment in testing of safety-critical embedded software. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–11.
[14] Gordon Fraser, Matt Staats, Phil McMinn, Andrea Arcuri, and Frank Padberg. 2013. Does automated white-box test generation really help software testers?. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. 291–301.
[15] Omar S Gómez, Karen Cortés-Verdín, and César J Pardo. 2017. Efficiency of software testing techniques: A controlled experiment replication and network meta-analysis. *e-Informatica Software Engineering Journal* 11, 1 (2017).
[16] Maria Fernanda Granda. 2014. An experiment design for validating a test case generation strategy from requirements models. In *2014 IEEE 4th international workshop on empirical requirements engineering (EmpiRE)*. IEEE, 44–47.
[17] Giovanni Grano, Simone Scalabrino, Harald C Gall, and Rocco Oliveto. 2018. An empirical investigation on the readability of manual and generated test cases. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 348–3483.
[18] M Harrer, P Cuijpers, TA Furukawa, and DD Ebert. 2019. Doing meta-analysis in R: a hands-on guide. *PROTECT Lab Erlangen* (2019).
[19] Mark Hays, Jane Huffman Hayes, and Arne C Bathke. 2014. Validation of Software Testing Experiments: A Meta-Analysis of ICST 2013. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE, 333–342.
[20] Sahitya Kakarla, Selina Momotaz, and Akbar Siami Namin. 2011. An evaluation of mutation and data-flow testing: A meta-analysis. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 366–375.
[21] G Kapfhammer. 2004. The Computer Science Handbook, chapter Software Testing.
[22] Barbara Kitchenham, Pearl Brereton, and David Budgen. 2010. The educational value of mapping studies of software engineering literature. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 589–598.
[23] Barbara A Kitchenham. 2004. Systematic reviews. In *10th International Symposium on Software Metrics, 2004. Proceedings*. IEEE, xii–xii.
[24] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. 2015. *Evidence-based software engineering and systematic reviews*. Vol. 4. CRC press.
[25] Pavneet Singh Kochhar, Tegawendé F Bissyandé, David Lo, and Lingxiao Jiang. 2013. An empirical study of adoption of software testing in open source projects. In *2013 13th International Conference on Quality Software*. IEEE, 103–112.
[26] Jeshua S Kracht, Jacob Z Petrovic, and Kristen R Walcott-Justice. 2014. Empirically evaluating the quality of automatically generated and manually written test suites. In *2014 14th International Conference on Quality Software*. IEEE, 256–265.
[27] Rogene Lacanienta, Shingo Takada, Haruto Tanno, and Morihide Oinuma. 2014. Test scenario generation for web application based on past test artifacts. *IEICE TRANSACTIONS on Information and Systems* 97, 5 (2014), 1109–1118.
[28] Nan Li, Anthony Escalona, and Tariq Kamal. 2016. Skyfire: Model-based testing with cucumber. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 393–400.
[29] Nan Li and Jeff Offutt. 2015. A test automation language framework for behavioral models. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 1–10.
[30] Arthur Marques, Franklin Ramalho, and Wilkerson L Andrade. 2014. Comparing model-based testing with traditional testing strategies: An empirical study. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 264–273.
[31] Phil McMinn. 2004. Search-based software test data generation: a survey. *Software testing, Verification and reliability* 14, 2 (2004), 105–156.
[32] James Miller. 2000. Applying meta-analytical procedures to software engineering experiments. *Journal of Systems and Software* 54, 1 (2000), 29–39.
[33] Audris Mockus, Nachiappan Nagappan, and Trung T Dinh-Trong. 2009. Test coverage and post-verification defects: A multiple case study. In *2009 3rd international symposium on empirical software engineering and measurement*. IEEE, 291–301.
[34] Glenford J Myers, Tom Badgett, Todd M Thomas, and Corey Sandler. 2004. *The art of software testing*. Vol. 2. Wiley Online Library.
[35] Alexander Pretschner, Wolfgang Prenninger, Stefan Wagner, Christian Kühnel, Martin Baumgartner, Bernd Sostawa, Rüdiger Zölch, and Thomas Stauner. 2005. One evaluation of model-based testing and its automation. In *Proceedings of the 27th international conference on Software engineering*. 392–401.
[36] Rudolf Ramler, Dietmar Winkler, and Martina Schmidt. 2012. Random test case generation and manual unit testing: Substitute or complement in retrofitting tests for legacy code?. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 286–293.
[37] Rudolf Ramler, Klaus Wolfmaier, and Theodorich Kopetzky. 2013. A replicated study on random test case generation and manual unit testing: How many bugs do professional developers find?. In *2013 IEEE 37th Annual Computer Software and Applications Conference*. IEEE, 484–491.
[38] Mark Rodgers, Amanda Sowden, Mark Petticrew, Lisa Arai, Helen Roberts, Nicky Britten, and Jennie Popay. 2009. Testing methodological guidance on the conduct of narrative synthesis in systematic reviews: effectiveness of interventions to promote smoke alarm ownership and function. *Evaluation* 15, 1 (2009), 49–73.
[39] Christoph Schulze, Dharmalingam Ganesan, Mikael Lindvall, Rance Cleaveland, and Daniel Goldman. 2014. Assessing model-based testing: an empirical study conducted in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 135–144.
[40] Guido Schwarzer, James R Carpenter, and Gerta Rücker. 2015. Multivariate meta-analysis. In *Meta-Analysis with R*. Springer, 165–185.

[41] Domenico Serra, Giovanni Grano, Fabio Palomba, Filomena Ferrucci, Harald C Gall, and Alberto Bacchelli. 2019. On the effectiveness of manual and automatic unit test generation: ten years later. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 121–125.

[42] Mario Linares Vasquez, Carlos Bernal-Cárdenas, Kevin Moran, and Denys Poshyvanyk. 2018. How do Developers Test Android Applications? *arXiv preprint arXiv:1801.06268* (2018).

[43] Wolfgang Viechtbauer. 2007. Confidence intervals for the amount of heterogeneity in meta-analysis. *Statistics in medicine* 26, 1 (2007), 37–52.

[44] M Vizard. 2013. App testing now consumes a quarter of IT budget. *CIO Insight* (2013).

[45] Xiaoyin Wang, Lingming Zhang, and Philip Tanofsky. 2015. Experience report: How is dynamic symbolic execution different from manual testing? a study on klee. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 199–210.

[46] Xiaojing Zhang, Haruto Tanno, and Takashi Hoshino. 2011. Introducing test case derivation techniques into traditional software development: Obstacles and potentialities. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 559–560.