Software Testing Effort Estimation and Related Problems: A Systematic Literature Review

ILONA BLUEMKE and AGNIESZKA MALANOWSKA, Warsaw University of Technology, Institute of Computer Science, Warsaw, Poland

Although testing effort estimation is a very important task in software project management, it is rarely described in the literature. There are many difficulties in finding any useful methods or tools for this purpose. Solutions to many other problems related to testing effort calculation are published much more often. There is also no research focusing on both testing effort estimation and all related areas of software engineering. To fill this gap, we performed a systematic literature review on both questions. Although our primary objective was to find some tools or implementable metods for test effort estimation, we have quickly discovered many other interesting topics related to the main one. The main contribution of this work is the presentation of the testing effort estimation task in a very wide context, indicating the relations with other research fields. This systematic literature review presents a detailed overview of testing effort estimation task, including challenges and approaches to automating it and the solutions proposed in the literature. It also exhaustively investigates related research topics, classifying publications that can be found in connection to the testing effort according to seven criteria formulated on the basis of our research questions. We present here both synthesis of our finding and the deep analysis of the stated research problems.

CCS Concepts: • **Software and its engineering → Software testing and debugging**; • **General and reference → Surveys and overviews**; • **Software and its engineering →** *Software reliability*;

Additional Key Words and Phrases: Testing effort, testing effort estimation, testing effort estimation-related problems, systematic literature review

ACM Reference format:
Ilona Bluemke and Agnieszka Malanowska. 2021. Software Testing Effort Estimation and Related Problems: A Systematic Literature Review. *ACM Comput. Surv.* 54, 3, Article 53 (April 2021), 38 pages.
https://doi.org/10.1145/3442694

1 INTRODUCTION

Despite the fact that testing effort estimation seems to be one of the most important tasks in software project management, it is surprisingly difficult to find any systematic method of such estimation. While searching for testing effort estimation methods, it is more likely to obtain methods for related tasks than for the main one.

Authors' addresses: I. Bluemke and A. Malanowska, Warsaw University of Technology, Institute of Computer Science, Warsaw, Poland; emails: {Ilona.Bluemke, Agnieszka.Malanowska}@pw.edu.pl.

0360-0300/2021/04-ART53 $15.00
https://doi.org/10.1145/3442694

Our interest in the testing effort estimation topic started in 2016. The primary objective was to identify existing tools supporting such process or implementable methods solving this task. It quickly appeared that searching for testing effort estimation-related literature is very challenging, as this problem is rarely considered separately from other software development estimation activities. Our preliminary research indicated the gap in the literature on software testing effort estimation, especially in its methods and tool support. We believe that this is a really meaningful problem from both scientific and practical points of view, and that motivated us to focus our research on this topic.

In this article, we describe the results of our systematic literature review (SLR) on testing effort estimation. It contains not only the solutions of that task, but also many results obtained in related areas. The main objective of this work is to identify existing methods and tools of software testing effort estimation and their usage in academia and industry. The other significant contribution of this article is our own categorization of many other solutions discovered during the research and presentation of the testing effort estimation task in a much broader context than in the other existing papers, including other SLRs (even SLRs published after the beginning of our research). None of the other publications we are aware of discuss the areas related to the testing effort estimation. Moreover, we present here not only quantitative results, but also their descriptive synthesis and the deep analysis of the findings, as well as the concrete answers to our research questions. We focused on the presentation of the results of our investigation (i.e., the identified methods and tools for testing effort estimation and the classification of other related research areas) and that is why their description forms the major part of the article. Our research was conducted similarly to the guidelines provided by Kitchenham [91], although some steps were performed informally, a bit differently or are not described here for brevity (details in Section 2). Let us also highlight the fact that out of **173** papers finally accepted for this SLR, 83 were published in 2016 or later, which means that 48% of the described solutions are less than five years old. Moreover, the accepted articles were selected from 309 primarily analyzed papers, which in turn were chosen from thousands of search results.

The article is organized in the following manner: The description of our research process, including the formulated research questions and the methodology of the survey, is presented in Section 2. The results of our SLR are exhaustively illustrated in Section 3, which contains also their critical discussion and the comparison with related works. In Section 4, we indicate some issues influencing the findings of our literature review. Section 5 concludes this article.

2 RESEARCH PROCESS

Our research on software testing effort estimation began in September 2016 and we have been working on it since that time. The preliminary results of our research were described (in Polish) in References [105, 106]. As those works are not published, we present here the significantly updated version.

The construction of this SLR was inspired by the construction of other literature reviews we had come across, e.g., References [3, 17, 47, 86]. In majority it is also compliant with the well-established SLR guidelines proposed by Kitchenham [91]. In fact, we followed as many steps of systematic SLR preparation rules as were possible, but the usage of some of them is not described in this article in detail for brevity. We did our best efforts to document the whole research process and below we present its most crucial details: research questions, description of the search process, including sources, keywords and access dates, inclusion and exclusion criteria, and the method of data extraction. Both quantitative and descriptive synthesis of the results are also presented in the article, as well as the analysis of the findings and the discussion, which can be seen in Section 3. We have also identified the threats to validity of the results (Section 4). Despite that fact, some of

the SLR steps proposed in Reference [91] were performed rather informally in this case. It applies mainly to the study quality assessment phase, as described in Reference [91]. This stage is not described here for two reasons. First, the initial quality assessment was done informally, based on the title and abstract relevance and their language quality. Second, we intentionally did not exclude some papers of poor quality regarding the topic of testing effort estimation to indicate the problems with obtaining high-quality resuls and to point out the low-quality publications and their authors. The critical discussion of them is presented in Section 3.2.1.

The main aim of our SLR was to identify existing methods of software testing effort estimation. Especially, we were interested in methods satisfying the following criteria:

(1) applicable in the early stage of the software development life cycle;
(2) possible to be automated (especially to be implemented in some programming language);
(3) independent of the used programming paradigm.

To achieve our goal, we have formulated research questions presented in Section 2.1. The methodology of our survey is described in detail in Section 2.2, while crucial research decisions are presented in Section 2.3.

2.1 Research Questions

In our first attempt to the literature review, we have formulated four general research questions:

RQ 1. Which methods of software testing effort estimation have been proposed in the literature?
RQ 2. Which automatic tools are available for testing effort estimation?
RQ 3. What are the measures of testing effort?
RQ 4. Which research topics are related to testing effort estimation?

It turned out that it is very difficult to find papers regarding this task as a separate one. It was also hard to find any methods or tools that could be used for testing effort estimation. There was an obvious gap in the literature. While searching for the articles on testing effort estimation, we have come across many other interesting papers on related topics and we decided to include them in the results of our literature review for two reasons. First, it can help in proving that there is a lack of publications about testing effort estimation itself. It can also show which topics are more common among the researchers. Second, it is an opportunity to present the interesting solutions from related areas. As far as we are concerned, there are no works covering such wide scope of topics, i.e., both testing effort estimation approaches and the diversity of related fields. Consequently, we have also decided to classify all retrieved papers. To do that, we have stated five more research questions.

RQ 5. What types of publications can be found in connection to the testing effort? Which solutions do they describe and how are they evaluated?
RQ 6. What is the main aim of the papers that can be found while searching for testing effort estimation methods? (Related to RQ 4)
RQ 7. Which types of methods are used in the area related to testing effort estimation?
RQ 8. What are the limitations of usage of the found methods?
RQ 9. What input data are required in case of the found methods?

2.2 Methodology

Our literature review was performed in three stages. The first part was conducted in September and October 2016 and its results are described in Reference [105] (in Polish). Then, the second part of the literature review took place in October 2018. Only the new papers (published in 2016–2018)

Table 1. Sources of Papers

Source	Number of papers
ACM Digital Library	33
IEEE Xplore Digital Library	53
Science Direct	29
Springer Link	26
Other (Internet)	32

were included during the second version of the literature review. The results of both parts of the survey are presented in Reference [106] (as we mentioned earlier, it was written in Polish and is not published). In the third part of the survey, prepared in June 2019, we wanted to ensure that no relevant papers had been omitted and to extend our results with the works published in the last year. This time we were searching for articles published since 2018.

To search for papers about testing effort estimation, we have used four full-text databases:

(1) ACM Digital Library,
(2) IEEE Xplore Digital Library,
(3) Science Direct, and
(4) Springer Link.

The search process was based on two search strings: "**testing effort**" and "**testing estimation**." Additionally, in the second and third stages of the survey, we also searched for relevant papers on the Internet, using Google search engine. The search on the Internet was conducted a little differently than in the databases. There was no restriction on date of publication and only one search term, "**testing effort estimation**," was used. That methodology was applied in both second and third stage of the literature review, and in both cases we have retrieved some different results.

The selection of papers was based on their titles and abstracts. We have scanned thousands of articles in total to select the most relevant ones. In the first stage of the survey, we have selected 69 papers from the full-text databases. The second phase of the literature review resulted in preliminary selection of 187 articles from the four databases. As the majority of them was not about testing effort estimation, but about related topics, we decided to include only one-third of papers on related topics in our results. We have preserved the proportions between all related topics. In the last part of our SLR, we have obtained 30 new articles, among which 11 were found in the databases and 19 had come from the Internet; some of them were excluded based on criteria described in next section. **The whole process resulted in selection of 173 papers for this SLR. Unfortunately, only 84 of them are related to the testing effort estimation problem.** As it was stated earlier, the rest of the publications are presented to indicate the difficulty in finding relevant solutions. The whole process of the SLR is depicted in Figure 1 and the number of papers selected from particular sources is presented in Table 1.

2.2.1 Inclusion and Exclusion Criteria. Selection of articles for this literature review was based on the search terms and dates of publications described in the previous section. In the first two phases of review, we have selected not only papers relevant for the topic of testing effort estimation, but also many papers concerning related topics.

There was also one more inclusion criterion. Sometimes while searching for papers on the Internet, we have found only abstracts, but full texts have not been freely available. In such cases, we were searching the Internet for the full texts of papers separately, providing the title and/or

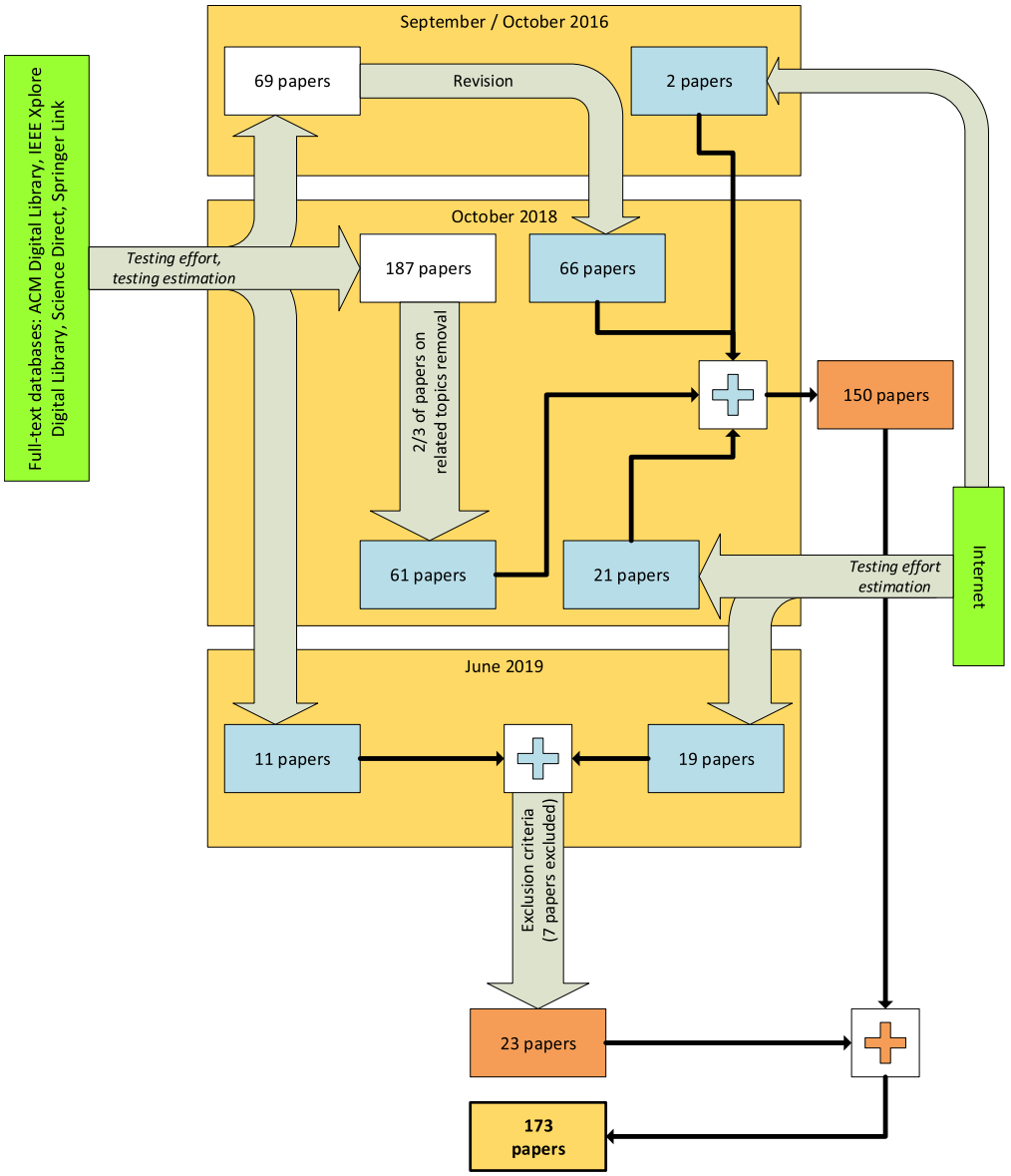

Fig. 1. Schema of the research process.

the author's name as keywords. Sometimes such search led to other interesting papers on testing effort estimation and we have included such additional results in our review.

Unfortunately, we also had to exclude some articles from the survey. First, we excluded papers not written in English. Second, we had to exclude the results that were not adequately bibliographically described and about which we were not able to check whether they had been published or who was the author. It happened especially with some results obtained from the Internet. Third, we obviously had to exclude the papers to which we did not have the full access.

2.2.2 Data Extraction. The process of extracting interesting data from the selected relevant papers was conducted a bit differently than defined by Kitchenham [91], which is caused by the nature of our research. We wanted to answer qualitative, not quantitative, research questions, and the extracted data were rather descriptive than numerical. We defined the categories of answers for research questions RQ3–RQ9 on the basis of the preliminary reading of some abstracts, and then we either assigned the papers to the existing categories or created the new one. Obviously, it required a few iterations of data extraction. The rest of research questions required concrete information about proposed methods (RQ1) or tools (RQ2) for testing effort estimation so information was retrieved for each relevant publication. All possible bibliographical data was also obtained for the relevant papers, including the title, authors, source, and location in the source publication, numerical identifiers of publications such as ISSNs, ISBNs, and DOIs and the URLs of the websites with access dates, depending on the type of the source.

2.3 Crucial Decisions

Although we had to make many decisions during the whole research process, as can be seen in the previous description, two of them seem to be the most important and have the significant impact on the final form of the survey. The first major decision was to extend the scope of our interest so it covers not only testing effort estimation problem, but also includes all other topics identified in the database search. The main reasons for that are the facts that we have found very small number of papers on testing effort estimation and that there is really no research presenting this task in a broader context. We did not restrict the scope of the related tasks, as long as they were in the domain of software testing or software engineering. It was done on purpose—we wanted to identify all problems related to testing effort estimation, not to predefine some subset of them. The second crucial choice concerned the formulation of the various classification types, according to which all papers (including publications on related topics) would be analyzed (i.e., the choice of interesting aspects of the selected publications), and the definition of categories for each classification type. It was determined iteratively—the preliminary scanning through the abstracts gave us the basic classification criteria and particular categories, which were completed after the careful analysis. As can be concluded, the most crucial decisions we made were to add the second set of research questions (RQ5–RQ9) and to formulate them, as well as determine the level of aggregation of the answers to them.

3 RESULTS AND DISCUSSION

As mentioned earlier, this study describes the results of the analysis of **173** papers on testing effort estimation and related problems. We answer our research questions in the reverse order, beginning with more general results and then going into details. First, in Section 3.1, the classification of all selected papers is given, in which we answer research questions RQ5–RQ9. Then, the discussion only about papers relevant for testing effort estimation is conducted in Section 3.2. It contains the answers to the research questions RQ1–RQ4.

3.1 Classification of Selected Papers

First, let us consider all papers selected for the survey, not only those on testing effort itself, but also those on related problems. They were classified according to the research questions RQ5–RQ9 presented in Section 2.1. All possible categories of one classification type were assigned to the paper, meaning that the same article can be included in several categories of a given classification type or that the article may not be considered in the particular classification type (i.e., the sum of papers assigned to all categories of a given classification type does not have to be equal to the total number of analyzed publications).

3.1.1 RQ 5. What Types of Publications Can Be Found in Connection to the Testing Effort? Which Solutions Do They Describe and How Are They Evaluated? It appears that three general types of publications can be observed:

- descriptions of new solutions,
- evaluations of solutions, and
- literature reviews.

Authors of 135 papers claim that they describe "new" or "novel" solutions. Fortunately, majority of them are somehow evaluated, although the quality of the validation depicted by different authors varies. The proposition of the new solution and its evaluation is described in **83** papers, namely, References [1, 2, 5, 7, 9, 11, 14, 18, 19, 24, 30, 31, 34, 35, 40, 45, 51, 60, 64, 65, 70, 75, 78, 82, 84, 85, 88, 93, 95, 98, 99, 101, 104, 110, 116, 120, 122, 124, 126, 132, 134, 137, 139, 149, 150, 152, 160, 162, 167, 169, 171, 173, 178, 181, 183, 184].

In 17 cases, the proposed novel solution is compared with the existing ones, to convince the reader that introduction of the new method is necessary. Such comparisons can be seen in References [7, 20, 23, 50, 53, 72, 89, 123, 125, 130, 134, 138, 145, 147]. It means that some kind of evaluation of the usefulness of the proposed novel solutions can be found in 95 articles.

Some authors provide an example of usage of their new solution, but such examples are insufficient to prove its applicability. That is the case in 14 papers, i.e., References [25, 28, 32, 59, 71, 73, 90, 117, 125, 128, 135, 141, 161, 172], although in Reference [125] the example of usage is accompanied by the comparison with other methods. However, the description of the new solution is sometimes preceded by the review of other existing solutions. Such construction of the paper is prepared to indicate the necessity of introduction of the new method [18, 152] or to derive the new method on the basis of the old ones [48, 182]. Only the first two of those papers [18, 152] contain the evaluation of the solution. There are also 24 papers containing only the description of the proposed novel solution whitout any evaluation, example, or review of other methods. It can be seen in References [10, 27, 36, 39, 46, 54, 57, 61, 63, 80, 81, 92, 94, 100, 102, 108, 118, 142, 168, 170, 174, 176, 179].

We can also observe that more than one-third of all novel solutions (i.e., 51 out of 135) are derived from other already existing methods. We consider here only specific methods proposed by other authors, not the popular statistical methods or machine learning algorithms. In the major number of 44 cases (that is, in References [2, 5, 7, 8, 9, 25, 26, 28, 36, 39, 50, 54, 57, 63, 70, 73, 75, 76, 82, 85, 94, 98, 99, 102, 117, 120, 125, 132, 135, 155, 161, 165, 167, 172, 173, 179, 182]), the new solutions are developed on the basis of propositions of other authors. It also happens that the authors develop their own work in the subsequent publications, such as in References [12, 16, 30, 31, 61, 154, 156, 166].

Not many publications focus on the task of evaluation of existing solutions. Four subgroups of such papers can be distinguished. We have found 2 papers that presented the evaluation of the method proposed earlier by the same authors. That is the case in the works by Badri, Toure, and Lamontagne [21, 166]. In those publications, they evaluate their previously proposed metric, Quality Assurance Indicator, in the context of predicting levels of testing effort in the unit tests [21, 166]. They use different machine learning algorithms and compare the results with those obtained for other known metrics. We observed that more often the existing methods are evaluated by other researchers, not by their authors. Such evaluations seem to be more reliable and trustworthy, as they have been conducted by the independent scientists. The results of those evaluations can be found in 6 papers [68, 69, 121, 148, 180, 185]. The other method of verification of the applicability of the solution is an empirical comparison with other existing ones. This is the most frequently used way of evaluation identified during this survey, as it can be found in 14 papers, i.e., References [4, 37, 49, 50, 64, 69, 96, 103, 111, 137, 140, 143, 148, 177]. Finally, only one publication [129] provides an example of usage of some existing solution.

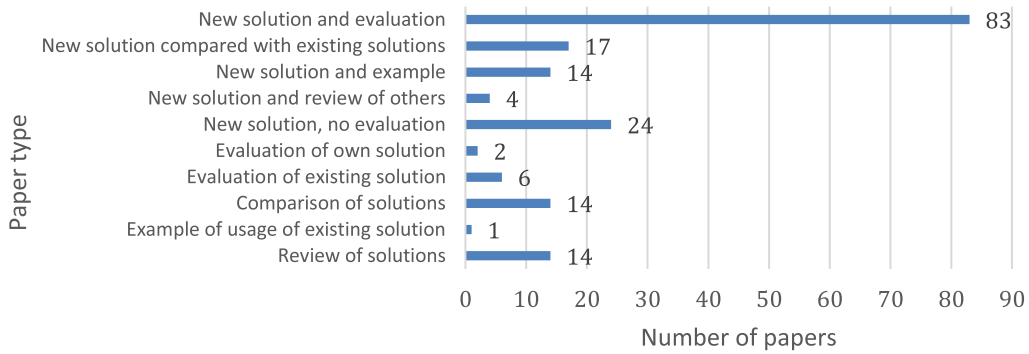

Fig. 2. Distribution of paper types.

The last group of papers consists of 14 literature reviews. Although such works seem to be extraordinarily precious in the context of searching for concrete implementable methods, almost half of them describe solutions for the related tasks. Garousi and Mäntylä [55] have analyzed 101 literature reviews on software testing. Some researchers explore the topic of test effort reduction. Bareja and Singhal [22] describe five artificial intelligence (AI) methods that can be used for such reduction, while Elberzhager et al. [47] present the results of their systematic mapping on testing effort reduction methods. Ghiduk et al. in their SLR [58] consider the topic of higher-order mutation testing and claim that such approach can result in testing effort reduction, particularly in comparison with first-order mutation testing. Hassan et al. [67] focus on the problem of testability and software performance, while Saeed et al. [140] survey the existing approaches for effort estimation for the whole software development, not particularly software testing.

We found eight literature reviews of software testing effort estimation but it seems to us that they are not very helpful while searching for a particular method. For example, Pinkster et al. [131] describe seven methods of testing effort estimation, but those approaches require the knowledge of an experienced expert and sometimes also use metrics counted for already completed software projects. Such solutions obviously cannot be automated. Kumari et al. [97] enumerate 10 methods of testing effort estimation, but they describe only two of them, i.e., Test Point Analysis (TPA) [168] and an adaptation of Use Case Points (UCP) [117]. Kafle [79] claims that it is very difficult to find papers on testing effort or cost estimation. His literature review is accompanied with a case study, and the results of the review itself are very generally described and are not really helpful in estimation methods identification, as he does not point out any specific publications. He observes that testing effort is often estimated as a part of the effort of the whole software development.

More valuable literature reviews on testing effort estimation have been conducted by Adali et al. [3], Araujo and Matieli [17], and Jayakumar and Abran [74]. Furthermore, Kaur and Kaur [86, 87] explore the problem of effort estimation for mobile applications. In their paper [87] they are looking for both development and testing effort techniques for traditional and mobile software. In Reference [86] they have combined the SLR of testing effort estimation methods for mobile applications with an industrial survey. It means that some parts of their results seem to be relevant for our research. However, none of the identified literature reviews cover as wide range of topics as this SLR. The detailed description of software testing effort estimation methods, including the results of the mentioned meaningful literature reviews, is presented in Section 3.2. The distribution of different types of publications identified during our survey is presented in Figure 2.

We have also identified three groups of solutions described in the selected papers. The vast majority of publications present some methods or models, either theoretical or practical, that are

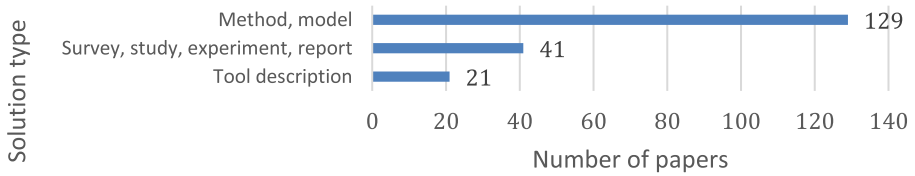

Fig. 3. Distribution of solution types.

usually suited for some specific problem. Specific methods and models can be found in 129 articles, i.e., References [1, 2, 5, 7, 14, 18, 20, 23, 28, 30, 36, 39, 46, 48, 50, 53, 57, 60, 61, 63, 65, 71, 73, 75, 78, 80, 82, 84, 85, 88, 90, 92, 94, 95, 98, 102, 104, 108, 110, 114, 116, 118, 120, 122, 123, 125, 126, 128, 135, 137, 138, 141, 142, 145, 147, 149, 150, 152, 161, 163, 165, 167, 176, 178, 179, 181, 184]. Further, we distinguish surveys, studies, experiments, and reports based on them as a second category of solutions. Such elements can be found in 41 publications, namely, References [3, 4, 6, 21, 24, 37, 41, 50, 53, 56, 57, 59, 66, 68, 70, 75, 77, 79, 86, 88, 92, 96, 103, 109, 110, 119, 121, 133, 140, 143, 144, 148, 151, 154, 166, 170, 173, 177, 180, 185].

The most interesting—although the smallest—group of solutions are descriptions of some tools that implement some method. The main objective of 16 publications is to present a software prepared by the authors. That is the case in References [10, 16, 27, 28, 54, 71, 73, 93, 115, 124, 133, 139, 153, 162, 169]. However, in References [24, 30, 31, 40, 78], we can only find an information that the proposed method is accompanied with a specific tool that automates its usage. Sahoo and Mohanty [142] describe the "proposed implementation" of testing effort prediction method. Although it seems very promising, the prepared tool is not presented there. During our search process, we have come across subsequent publications of those authors, but only the abstract was available. We asked for the full text of the paper, but we did not get any response. Consequently, we were not able to detect whether the "proposed" tool has already been implemented. It means that, to the best of our knowledge, their proposition cannot be categorized as a tool description. Furthermore, the authors of Reference [142] are also co-authors of Reference [141], which is very similar to the former one. In fact, in Reference [141] Sahoo et al. describe their approach for system test effort estimation from use cases diagrams only and the manner of use case assessment is the same as in Reference [142]. They have implemented in Reference [141] two steps of their approach, which consist of use case model creation in a CASE tool, export of the model to the XMI format, XMI file (i.e., use case model) parsing and storing some data in the repository. The further steps of their proposition, which need to be executed to estimate the effort of system testing, have not been implemented. As a consequence, this article cannot be regarded as a tool description. Unfortunately, only 10 papers describe the existing tools more or less connected to the question of testing effort estimation. They are described in detail in Section 3.2.2. The distribution of all types of solutions identified in the selected papers is presented in Figure 3.

3.1.2 RQ 6. What Is the Main Aim of Papers That Can Be Found while Searching for Testing Effort Estimation Methods? However, the primary goal of our literature review was to find papers describing testing effort estimation methods—only **84 of total 173** publications satisfy that condition. It seems to be a small percentage, especially if we keep in mind that those 173 articles were selected from many more others; it illustrates the difficulty in finding relevant sources. A lot of papers focus on other topics, somehow related to testing effort, and below we present some of them.

Some of the papers we come across cover wider scope than testing effort estimation. We have identified two categories of such papers.

First, the subject of software development effort estimation (i.e., estimation of effort for the whole development process, not just for testing phase), or at least estimation for the wider scope than only testing, is more exhaustively explored, for instance in the following 13 papers: [23, 35, 66, 70, 77, 87, 111, 132, 140, 143, 147, 160, 170]. Moreover, some of the mentioned approaches are very specific and can be used only for particular types of systems. It can be observed in the work of Ramacharan and Rao [132]. They propose the method for software development effort estimation suited for Global Software Development (GSD) projects that are developed by people from different cultures [132]. Further, Salmanoglu et al. [143] focus only on the agile projects, while Calzolari et al. [35] try to estimate effort neither for the whole software development nor for the testing phase—they model the effort for the testing and maintenance phases [35], but that approach cannot be used for estimation of testing effort itself. Jodpimai et al. [77] describe the method of re-estimation of the effort during the project on the basis of the amount of effort from the previous phases. Mensah et al. [111] propose the model with two output values, which they call "duplex output." The first value in their approach is the estimated amount of software development effort, and the second—the level of effort, measured as low, moderate, or high [111]. Wang et al. [170] aim at describing the time distribution of effort during the particular months of project, but in this case, software development process is divided into months, not into phases. Finally, Kaur and Kaur [87] investigate both the software development and testing effort estimation methods.

To the second category of papers that cover the wider scope than only testing effort estimation belong also Software Reliability Growth Models (SRGMs). Such models depict the dependency between the number of defects in the software and the time spent for testing [78], what can also be interpreted as the improvement of the system reliability in time [89]. We have selected 16 exemplary publications belonging to the domain of software reliability growth modeling, i.e., References [69, 78, 81, 82, 89, 99, 103, 121, 126, 128, 129, 175, 176, 178, 180, 185].

Much attention is paid in the literature to the different ways of test process enhancement or automation. The dominant topic among them is automatic tests generation. Ansari et al. [10] propose the method for test case generation using Natural Language Processing (NLP) techniques. Sneed [153] prepared the tool that generates test cases on the basis of different requirements specification documents written in English or German. Further, UML model is used in automatic test case generation method proposed by Elallaoui et al. [46], but in this case, only agile projects using Scrum methodology are considered. The tool developed by Moketar et al. [115] generates so-called "abstract test components" based on the simplified models of use cases and user interface (called, respectively, Essential Use Cases and Essential User Interface [115]). Subsequently, Křikava and Vitek [93] prepared the tool that creates unit tests for R programs on the basis of their execution traces, while the software of Parizi et al. [124] generates tests for aspect-oriented programs written in AspectJ.

Automatic test generation is not only visible in the developed tools, but also in a more scientific research. Ferrer et al. [51] try to define a measure describing the difficulty of automatic test case generation. Endo and Simao [49] assess the methods of test suite generation on the basis of finite state machine models. Similar research is done by Holt et al. [68]—they investigate the effectiveness of test models based on the UML state machine diagrams and of test suites prepared from that models. Elghondakly et al. [48] also evaluate different methods of test case generation and propose their own solution. The reasonableness of automatic unit test generation is verified by Shamshiri et al. [144]. They have checked whether the benefits from the quicker test preparation are not less than the difficulty in understanding of tests created by the machine and the possible subsequent problems in the maintenance phase [144]. Their results indicate that the source of tests has no impact on the efficiency of the developers.

Furthermore, not only test generation can be automated. For example, Aman et al. [9] present the method for automatic recommendation of test cases that should be re-executed during regression tests. Their method takes into account the similarity between test cases. Surprisingly, their primary objective is not to reduce testing effort, but to ensure that all essential test cases will be executed during regression tests and none of them will be omitted. Bertolino et al. [24] investigate the subject of automatic determination of test paths that assure the desired level of coverage in branch testing. Ndem et al. [119] describe the way of selection of test data used during automated testing. Rosenfeld et al. [138] propose the method that can improve the execution of functional tests of Android mobile applications. Their solution is based on the observation that the generic test scenarios can be prepared for similar screens of applications and then they can be reused. Sadeghi et al. [139] also consider the question of Android mobile apps testing, but they focus on testing the software behavior for different combinations of permission settings.

There also exist approaches used to test prioritization, reduction, and selection. Test case prioritization aims at obtaining such order of test cases that the tests allowing to detect the highest number of faults are executed as the earliest [135, 164, 171]. Different methods of test case prioritization are described—for instance—in References [135, 154, 171, 173, 174]. Czibula et al. [42] focus on the task of setting class integration test order (CITO). The work of Flemström et al. [53] also covers integration testing, but they investigate the order of automation of manual test cases to maximally use the previously automated elements. Test reduction, also known as test minimization or test optimization, is in fact the decrease of the size of tests [61, 95] without the deterioration of their quality [122]. The removal of unnecessary test cases is considered by Kumar and Bhatia [95]. Palomo-Lozano et al. [122] use different methods to reduce the size of mutation tests, preserving the constant amount of code coverage. Furthermore, Groce et al. [61] try to lower not only the size, but also the semantic complexity of tests using ideas of normalization and generalization of tests. Khurana et al. [90] analyze the impact of changes in the system to estimate regression test effort, but meanwhile in their approach, the test suite is also reduced. However, test selection means that some subset of the original test suite is chosen for execution [137, 164]. In this case, test cases that do not improve the quality of the system should be rejected [164]. Test selection is most often conducted for regression testing phase, as can be seen in the examples of References [133, 169]. The authors of References [133, 137, 169] indicate that for regression test selection, primarily elements of the system affected by changes in the software should be considered. Rosenblum and Weyuker [137] investigate the effectiveness of different regression test selection methods, while Aranha and Borba [14] describe the method helping in selection of the test cases that are to be automated. There are also some approaches that combine some of the above techniques. Miranda and Bertolino [113] propose methods for prioritization, reduction, and selection of white-box tests in the context of systems developed with reused code. Tahvili et al. [164] consider prioritization and selection of integration tests in a dynamic way, using the results of previously executed tests.

Other approaches use fault-proneness determination. In those approaches, particular software unit is assigned to one of the two classes: fault-prone or non-fault prone parts [45, 96, 123]. Often, the whole modules are regarded as software units, as can be observed in References [45, 112, 123]. Although the main idea beyond such approaches is usually very similar, the details can differ a lot. Dhanajayan and Pillai [45] propose Bayesian classification for projects developed according to the spiral life cycle model. Koroglu et al. [92] use data about defects identified in the previous versions of the software to predict the fault-proneness of the new release. Moreover, they focus on the legacy software [92]. Kumar and Sureka [96] are, in turn, interested in prediction of defects resulting from the fact that the software is getting older, i.e., aging-related bugs. In their case, the main software units classified as fault- or non-fault-prone are files or classes. Yan et al. [177] also treat files as the main system units, but they aim at comparing the effectiveness of different

models, obtained with supervised and unsupervised learning algorithms. Interestingly, Mesquita et al. [112] present two versions of the method for fault-proneness determination. They introduce the reject option to their fault-proneness classifier, so in the doubtful cases, the expert knowledge can be used to prevent the problems resulting from misclassification [112]. Shippey et al. [149] investigate the relationship between abstract syntax trees (AST) of Java software and the existence of defects. Pandey and Goyal [123] go even further and propose the method not only to classify the modules as fault- or non-fault-prone, but also to rank them according to the level of fault-proneness. Moreover, Rathore and Kumar [134] developed a model that predicts the number of defects in particular software modules. Shihab et al. [148], in turn, consider the problem of finding location of bugs in the code, i.e., bug prediction, and compare the results obtained for different source-code metrics. Debroy and Wong [44] aim at estimating the number of test cases sufficient for fault detection, while Göldner [59] tries to estimate the cost of late identification of defects in the production environment.

Getting closer to the question of testing effort, we can observe that the main objective of many authors is to reduce (or minimize) testing effort. It is clearly stated in many papers, such as References [6, 10, 22, 40, 42, 46, 47, 53, 58, 61, 65, 92, 93, 95, 96, 101, 108, 115, 116, 119, 122, 124, 133, 138, 139, 169, 172, 174]. It is also easy to notice the general rule that all techniques used to enhance the testing process aim at test effort reduction. That rule applies for all methods of testing improvement mentioned above: automatic test generation [46, 115, 124], test prioritization [101, 174], selection [101, 133, 169], and reduction [122]. In none of those cases the amount of effort is estimated. Such specific value of effort, either exact or approximate, seems to be irrelevant; the attention is paid only to maximally reduce the effort, regardless of its original or resulting value. Kappler [80] strictly claims that the main objective of his solution is to speed up the execution of tests.

The next important group of publications consists of solutions of testing resource allocation problem. References [52, 104, 118] are particularly dedicated to that subject. Fiondella and Gokhale [52] clearly define that they aim at achieving particular level of software reliability with the smallest expenditures. We should also bear in mind that the optimal resource allocation is the main objective of software reliability growth modeling [69, 99, 118, 128, 129] and fault prediction [123, 177]. Again, in any of those papers the value of testing effort is not determined, because they focus on the identifications of parts of software that require more testing (in terms of time or human work).

Even among the mentioned 84 publications, somehow related to testing effort estimation problem, some subgroups can be distinguished. Badri et al. in References [19, 20] try to estimate the size of code of the test suites, while Badri, Toure, and Lamontagne in References [21, 166] estimate not the value, but the level of effort, considering only unit tests. Estimation of effort required only for test execution is quite common. The issue of test execution effort estimation is for example deeply explored by Aranha et al. in many papers, e.g., References [11, 14, 16], although in Reference [13] they focus on tests executed manually, while in Reference [14] they investigate the effort for both test execution and test automation. Estimation of manual test execution time is also considered by Tahvili et al. in References [162, 163]. Ashraf and Janjua [18] try to estimate the effort of test execution in agile projects that use extreme programming technology. Their approach is based on the requirements specification in the form of user stories. Zhu et al. in References [183, 184] also aim at estimating test execution effort using different methods. Silva et al. [150] describe their method of estimating execution effort of functional test cases. Finally, Thirasakthana and Kiattisin [165] try to improve the test execution effort estimation using Lean Six-Sigma methodology.

We have also identified publications that describe methods for estimation of other stages of the testing phase. Silva-de-Souza and Travassos [151] search for the factors that influence the effort

of test design and implementation, while Yang [179] proposes the method of estimation the effort of test automation.

Some of identified papers are rather indirectly related to the topic of testing effort estimation and although they belong to that thematic group, they do not provide any methods for such an estimation. Bhattacharyya and Malgazhdarov [27] describe the tool for determining the code coverage for the fixed time and budget of a project. Böhme [34] compares the software testing process to the discovery of plant species and claims that biological observations can be used to estimate the time required to complete the testing phase. Grano et al. [60] propose a new approach for the assessment of the effectiveness of test cases, instead of the mutation analysis. However, Kazerouni et al. [88] describe the metrics for evaluation of practices of testers in the context of incremental software development. They consider only testing effort spent in a one increment of work [88]. Marré and Bertolino [108] try to estimate the number of tests necessary to cover the assumed amount of code and, at the same time, to minimize that number as much as possible. Martins and Melo [109], in turn, aim at determining the number of test cases that have to be generated to assure the desired value of Modified Condition/Decision Coverage (MC/DC) metric. Periyasamy and Liu [130] propose the set of metrics that can help in testing effort estimation for object-oriented software. Unfortunately, they do not provide the estimation results in any of widely accepted measures, they only give some numeric values. As mentioned earlier, Sneed [153] claims that the usage of his tool for determination of the number of test conditions will improve the test effort estimation. Finally, Zapata-Jaramillo and Torres-Ricaurte [182] use the elements of the existing methods of testing effort estimation to develop a graphical, schematic representation of that process, but they do not describe any method of such an estimation.

Testing effort estimation methods often have some limitations. First, a lot of them are intended for the estimation of effort for some specific level of testing. Dawson [43] considers only unit testing effort, while Kumar and Beniwal [94] focus on unit and integration tests. The approaches of Lazić and Mastorakis [102] and Sahoo et al. [141, 142] cover only system testing, while the works of Jayakumar and Abran [75], Jiang et al. [76], and Silva et al. [150] can be applied only to functional testing. Finally, Abhilasha and Sharma [1] and Khurana et al. [90] try to estimate the effort of regression tests.

Second, the input data required for the estimation methods can cause some limitations in their usage. For example, solutions of Choudhary and Dhaka [39] and Nguyen et al. [120] need test cases as an input, Nageswaran's [117] method uses use cases, while approaches presented by Sharma and Kushwaha [145, 146] require Software Requirements Specification (SRS) document. Further, the methods of Kashyap et al. [84] and Kushwaha and Misra [100] are based on source code, while the solution described by Srivastava et al. [159] needs historical data.

Third, among the other limitations of the selected publications regarding testing effort estimation, one can also observe the limited scope of application or impossibility of implementation. Methods described by Pinkster et al. [131] are based on the experience of experts, which makes them unable to be fully automated. However, Cotroneo et al. [41] present approach for testing effort estimation for open source software, using real-time operating system as an example. Furthermore, Ahmad and Samat [5] aim at extracting the cost necessary for quality and testing activities from the cost of the whole project in case of the testing performed by outsourcing company. They have built a regression model based on the Malaysian governmental project data [5]. Finally, the solutions presented Bock et al. [30, 32] are suited for driver assistance systems that use data obtained by various sensors as an input—that is the case of an embedded system, which is beyond the scope of our interest.

As we stated before, our objective was to find methods that can be used in as wide scope as possible, early in the project development and to estimate the effort for the whole testing phase.

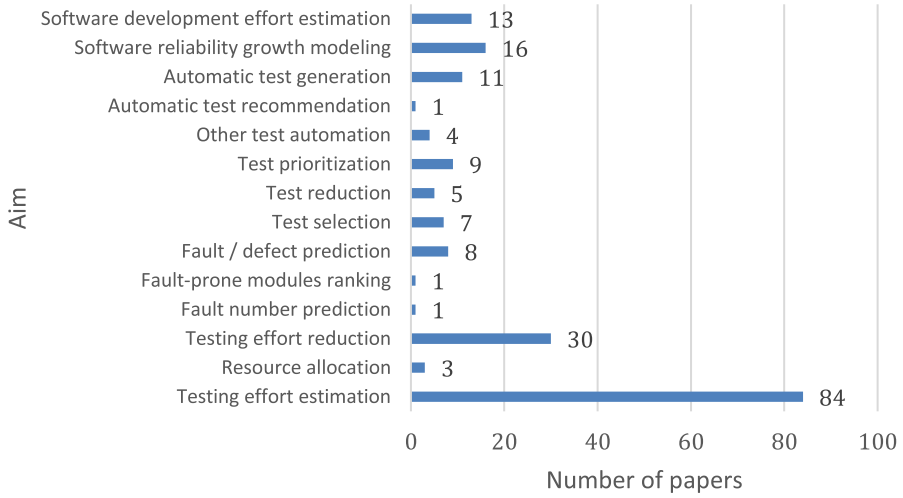

Fig. 4. Distribution of papers according to their main goal.

Consequently, none of the approaches described above can be applied in our case, as none of them satisfies all those conditions. The more detailed description of different software testing effort estimation methods is presented in Section 3.2. The distribution of main subjects of the selected papers is presented in Figure 4.

3.1.3 RQ 7. Which Types of Methods Are Used in the Area Related to Testing Effort Estimation? Two main groups of methods are used in the area related to testing effort estimation, i.e., AI methods and analytical or statistical methods. Unfortunately, as can be seen further, sometimes it is quite difficult to assign particular solutions to one of those categories.

The vast majority of all types of methods are various measures and metrics, and we have identified 47 papers that describe such kind of approach. Propositions of new metrics or solutions developed on the basis of existing measures are presented in References [1, 7, 11, 13, 19, 25, 26, 28, 36, 39, 50, 51, 56, 63, 66, 71, 73, 84, 88, 94, 98, 100, 102, 117, 120, 125, 130, 141, 142, 145, 147, 150, 152, 154, 159, 161, 168, 173, 179, 184]. Software Reliability Growth Models (SRGMs) can also be assigned to the group of analytical solutions, as well as metrics. Sixteen approaches related to software reliability growth modeling are listed in Section 3.1.2. Methods presented by Mensah et al. [110] and Palomo-Lozano et al. [122] are based on Integer Linear Programming (ILP) technique. Umar [167] models testing effort using Cobb-Douglas function, while Wang et al. [171] use probabilistic risk analysis to prioritize test cases. Bock et al. apply discrete-time Markov chain [30, 31] and other mathematical methods [32], while Thirasakthana and Kiattisin [165] use Lean Six-Sigma approach. The other analytical approaches can be noticed in References [59, 65, 75, 90, 104, 114, 170].

Approaches using source code analysis also can be assigned to the group of analytical methods, although not to the mathematical or statistical ones. Bertolino et al. [24] perform static analysis of the control flow graph, while Sadeghi et al. [139] explore both source code and test suites. Gupta and Jalote [64] apply mutation analysis, while Khurana et al. [90] analyze the source code to detect the impact of changes on the system. Our approach [28] also can be classified as an analytical one, as it investigates the relationships between UML model elements and, in addition, uses previously defined measures.

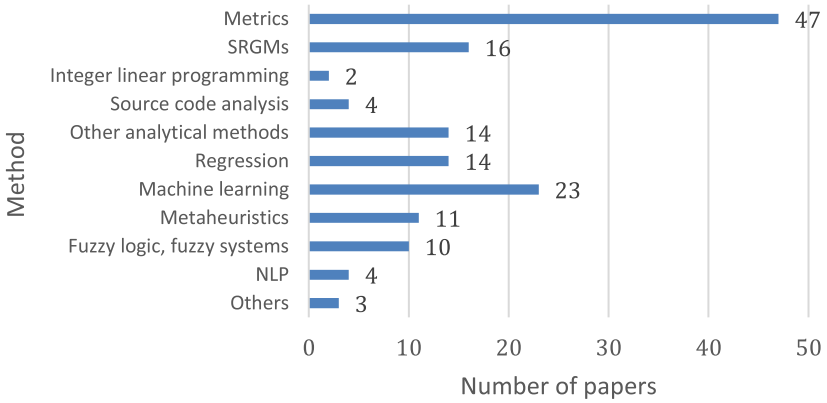

Fig. 5. Distribution of types of methods used in papers.

Solutions that use different types of regression method are the most difficult to be assigned to one of the two main groups. Obviously, regression is a statistical method; however, it is often used in machine learning process. Among the selected literature, examples of linear [5, 20, 75, 85, 114, 163], logistic [21, 41], and LASSO regression [27] can be found. Mensah et al. [111], Tahvili et al. [162] and Toure et al. [166] use some regression models in their works. Srivastava [156] uses fuzzy multiple linear regression. Regression is also used by Okamura et al. [121].

Diverse machine learning techniques can be assigned to the group of AI approaches. Classification algorithms are the dominant ones and can be found in References [20, 21, 41, 45, 60, 92, 111, 112, 123, 138, 166]. Clustering methods are used rarely—only in two publications [45, 95]. Authors of References [2, 43, 50, 110] build their solutions using artificial neural networks. Machine learning methods are also utilized in References [42, 70, 77, 96, 134, 149, 183]. The second important subgroup of AI techniques consists of metaheuristics. Particular attention is paid to Particle Swarm Optimization (PSO) and genetic algorithms. The former is used in References [8, 26, 160], while the latter—in References [101, 118]. There appear also many other types of metaheuristics, such as differential evolution [23], firefly and water wave algorithms [98], bat algorithm [157], cuckoo search method [159], and so on [116, 173].

Approaches based on various fuzzy techniques can be treated as AI methods, too. Choudhary et al. [39], Srivastava [155, 156] et al. [158], and Yadav and Dutta [174] use fuzzy logic, while Kumar and Bhatia [95] apply fuzzy clustering. Pandey and Goyal [123] take advantage of fuzzy inference system, while Suharjito et al. [160] exploit the adaptive neuro-fuzzy inference system (ANFIS). Other fuzzy methods are also used in References [63, 70]. The last subgroup of AI techniques contains solutions that process natural language (i.e., NLP solutions). We have identified four approaches of that kind, namely, References [10, 16, 153, 163]. Finally, methods used by Aman et al. [9]—that is, morphological analysis and Mahalanobis-Taguchi method—also can be classified as an AI approach, but they do not belong to any subgroups mentioned above.

There are some solutions that cannot be classified according to the previously defined criteria. That is the case with publications written by Pinkster et al. [131] and Kappler [80]. In the first one, methods based on expert knowledge are described, while the main aim of the second one is to detect and remove dependencies between tests to enable their parallel execution. The distribution of all kinds of methods that are used in the selected literature is presented in Figure 5.

3.1.4 RQ 8. What Are the Limitations of Usage of the Found Methods? A lot of solutions described in the literature cover only some particular type of software or project life cycle. Moreover, many

methods related to various aspects of software testing are limited to particular levels of testing. Here, we present the different categories of limitations of publications selected for this review.

Solutions for object-oriented software are the most often proposed ones. It is not surprising, as OOP is one of the most common programming paradigms nowadays. Authors of 13 papers, i.e., References [19–21, 28, 66, 84, 96, 130, 141, 142, 166, 172, 174], focus on such software. Only one example among all retrieved publications can be found for each of two other identified paradigms. The tool of Parizi et al. [124] is suited for aspect-oriented programs, while Harrison and Samaraweera [66], in addition to object-oriented software, explore also functional programs. Yang et al. [180] propose the method for web applications, while Palomo-Lozano et al. [122] and Silva-de-Souza and Travassos [151] present the solutions regarding web services. Cotroneo et al. [41] use their approach for the open source real-time operating system. Another method for open source software is prepared by Johri et al. [78].

Approaches prepared particularly for mobile applications also seem to be quite common. Methods and tools described in References [40, 85, 138, 139, 178] and References [86, 87] are dedicated to that type of the software. Moreover, the majority of described methods (i.e., References [40, 138, 139]) are suited for Android devices. However, solutions presented in References [18, 46, 87, 125, 143] are prepared especially for agile projects. Further, in References [132, 172] the question of distribution is addressed. However, it is done from very diverse perspectives, since Ramacharan and Rao [132] investigate the systems that are developed by geographically distributed programmers (that is, the problem of Global Software Development), while Wong et al. [172] consider distributed object-oriented systems (i.e., the system itself is distributed). Some researchers develop their solutions [5, 74, 125] in the context of outsourcing of the test activities. We can observe also some interest in the question of code reusability. Four articles [6, 40, 54, 113] describe solutions in the context of the potential reuse of some parts of software. Methods related to the task of code refactoring also appear in our search results. Such refactoring-oriented approaches are introduced by Morales et al. [116] and Wang et al. [169]. Some solutions are developed for a particular programming language. The proposition of Cleva, Farto, and Endo [40] is dedicated to tests written in Java; Křikava and Vitek [93] consider R programs testing; the tool of Parizi and Ghani [124] can be used for test generation for AspectJ software; while Shippey et al. [149] investigate systems developed in Java. Interestingly, some authors indicate that their solution is independent of the programming language [130] or the technology used [36], or even that its scope of application is very wide [101].

Several papers focus on very specific applications. Bhattacharyya and Malgazhdarov [27] assume the usage of symbolic execution tool. Bock et al. in subsequent papers [30–32] developed the approach for test effort estimation for driver assistance systems. The work of Gay et al. [56] considers avionic systems as an example of critical software, while Koroglu et al. [92] focus on the legacy systems. Nasar and Johri [118] build resource allocation model for the dynamic environment. The distribution of various limitations of usage of the solutions presented in selected papers is shown in Figure 6.

The types or levels of testing considered in particular works also lead to some limitations in the application of the found solutions. The majority of approaches focus on unit tests. Solutions presented in References [21, 43, 93, 119, 144, 166, 181] can be applied only to unit testing phase, while propositions described in References [42, 53, 164] are dedicated to integration tests. Kumar and Beniwal [94] propose approaches for both unit and integration testing levels. System testing is considered as a separate problem by Lachmann et al. [101], Lazić and Mastorakis [102], and Sahoo et al. [141, 142]. All levels of testing (i.e., unit, integration, system, and acceptance testing) are covered only by Veenendaal and Dekkers [168] in their TPA method. As a consequence of the TPA usage, all those levels are also considered in our approach [28] and its improvement [107, 29]. Regression testing is taken into account in 10 publications, i.e., References [1, 9, 16, 65, 90,

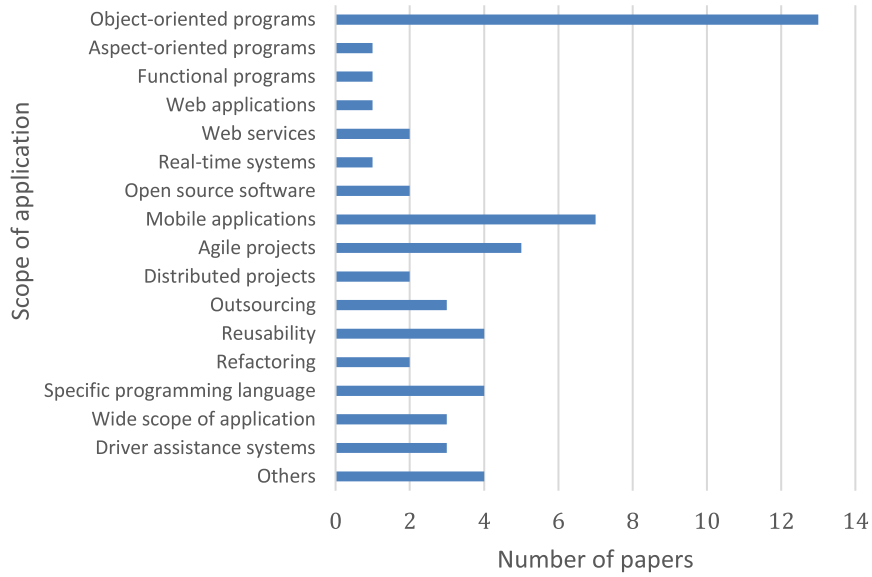

Fig. 6. Distribution of limited scopes of application of the solutions.

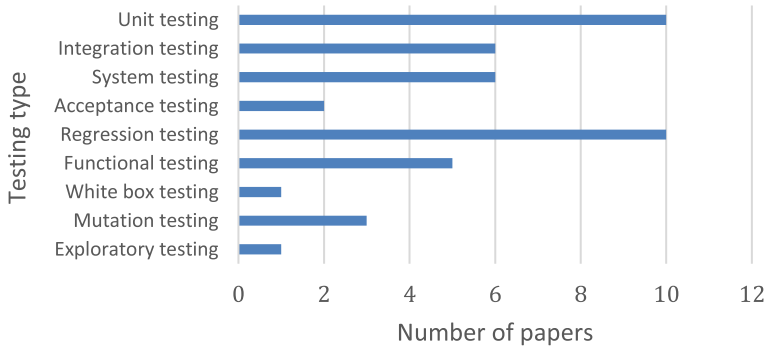

Fig. 7. Distribution of types of testing considered in solutions.

133, 137, 169, 171, 174]. Five papers [13, 75, 76, 138, 150] cover the functional testing question, while Reference [113] can be used for white-box tests. In three cases, namely, References [58, 64, 122], mutation testing is explored. Finally, Afzal et al. [4] dedicate their solution to exploratory testing. The distribution of different types and levels of testing considered in different solutions is presented in Figure 7.

3.1.5 RQ 9. What Input Data Are Required in Case of the Found Methods? There are several types of input data that are used in methods described in the selected set of papers. In majority, some kind of metrics or measures are used as input values. It can be observed in 36 papers: References [5, 8, 20, 21, 28, 32, 41, 43, 60, 66, 71, 73, 75, 85, 92, 96, 102, 112, 121, 123, 125, 131, 143, 148, 150, 154, 156, 160, 161, 165, 166, 168, 173, 174]. Particularly, metrics are often used as an input for machine learning algorithms (classification, clustering, regression, and so on, as can be seen in References [5, 20, 21, 41, 43, 60, 75, 85, 92, 96, 112, 121, 123, 156, 166]) or for meta-heuristics (in References [8, 160, 173]); see also Section 3.1.3. However, historical data about

completed projects are also often used. We can distinguish two cases of previous projects data usage. First, the historical data can be used for analysis execution, e.g., as an input for a tool or a variable in some method. Second, such data can be used to create dataset to which AI methods are applied. The former phenomenon can be observed in five papers. Pinkster et al. [131] describe metrics that require data about past projects, Kuo et al. [99] need such data to develop SRGM, while Veenendaal and Dekkers [168] and Yang [179] utilize only the value of productivity factor calculated on the basis of previous projects history, such as we [28] do. The latter approach is much more common. Historical data are often used in machine learning methods to build an estimation model (e.g., a classifier or a regression model), such as in References [2, 5, 27, 43, 45, 50, 70, 75, 77, 85, 92, 96, 110, 112, 114, 121, 123, 134, 138, 149, 156, 162, 163, 183]. Sometimes historical data are also used together with metaheuristics, as can be seen in References [23, 101, 157, 159].

There are also several approaches that employ some kind of requirements specification as an input. Often it is Software Requirements Specification (SRS) document. Ansari et al. [10] use it for automatic test case generation, Chaudhary and Yadav [36] propose SRS document usage in some approach related to the Test Case Point approach (see Section 3.2.1 for more explanation about issues with TCPA usage), while Sharma and Kushwaha treat SRS document as a basis for estimation of testing effort [145, 146] or total software development effort [147]. Srivastava et al. [155, 156, 158] and Bhattacharya et al. [26] also use SRS document for test effort estimation using different AI approaches. Ashraf and Janjua [18] estimate testing effort using user stories as the input data. Requirements specification is also used by Sneed [153] and Srikanth et al. [154]. The former approach needs them for test case generation; the latter for test prioritization. Sometimes also elements of the software design are treated as an input. It is usually some part of the UML model of the system [28, 46], e.g., class diagram [65, 90, 142] and/or use case diagram [65, 141, 142]. Use case specification is also used as an input in References [7, 19, 20, 25, 115, 117, 161, 184].

The other approaches rely on data obtained from various test artifacts. Aranha et al. [11, 13, 14, 16] and Tahvili et al. [162, 163] employ test specification for that purpose. Other researchers use test models, test cases, or test suites as input [12, 39, 40, 53, 64, 76, 120, 139, 174, 179]. In approaches presented by Gupta and Jalote [64] and Sadeghi et al. [139], input requires also source code of the software. Source code is also used as input by 10 other solutions, namely, References [27, 44, 60, 84, 88, 90, 100, 130, 149, 171]. Some researchers try to exploit the control flow graph as an input [24, 171], while the others use application screens and user interface [40, 138] for the same purpose. Unfortunately, all those type of input data are available very late in the software development life cycle, which can be a serious limitation of their usage.

Some approaches that can be found in the literature take advantage of quite unconventional types of input data. Křikava and Vitek [93] use information from program execution traces; the method presented by Kazerouni et al. [88] utilizes data obtained from commits; Kumar and Beniwal [94] apply the knowledge about the usage of test stubs and drivers; while Shippey et al. [149] predict the fault-proneness on the basis of abstract syntax tree of Java source code. The solution of Ferrer et al. [51] requires program representation in the form of Markov model, while the tool of Moketar et al. [115] needs simplified use case and user interface models. Further, the tool developed by Ramler et al. [133] uses the data about modifications introduced in the system and test coverage. Yang et al. [180] describe the approach that requires workload and failure data, while solutions presented by Bock et al. [30, 31] need information retrieved by the sensors. The distribution of all types of input data that were identified during our research is illustrated in Figure 8.

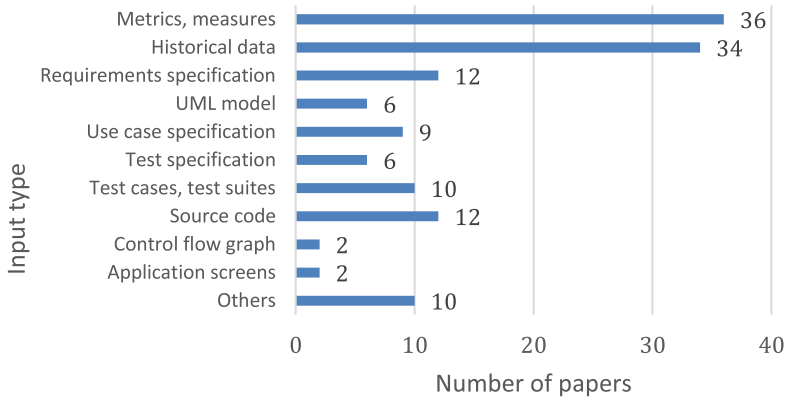

Fig. 8. Distribution of input data necessary for the solutions.

3.2 Approaches to Testing Effort Estimation

The main objective of our research was to identify useful, applicable, and implementable methods of software testing effort estimation. It turned out that there are only few approaches to this task. Here, we also describe the results of our investigation on tools and measures for testing effort and enumerate other topics related to that one. In the following subsections, we answer the research questions RQ1–RQ4 stated in Section 2.1.

3.2.1 RQ 1. Which Methods of Software Testing Effort Estimation Have Been Proposed in the Literature? During our SLR, we have identified 84 papers covering the topic of testing effort estimation. Although the absolute number of such papers may seem to be quite big, we should be aware that it is less than a half of all papers selected for this review and only a small fraction of all retrieved search results. Consequently, we believe that the number of valuable articles on testing effort estimation is relatively small in comparison to the number of publications obtained during the search for testing effort estimation method that turn out to cover related topics. Moreover, the number of valuable articles on testing effort estimation is less than 84, as we have also found some poor quality papers. We decided not to exclude them from the review, but to point out the untrustworthy publications and their authors. We have also observed that the names of authors of papers on testing effort estimation methods repeat, and there is a limited number of research groups interested in that task.

There are two most common analytical approaches to the testing effort estimation problem in the literature:

- Test Point Analysis (TPA) [168] proposed by Veenendaal and Dekkers in 1999 and
- Use Case Point (UCP) method introduced by Nageswaran [117] in 2001 (a modification of original UCP proposed by Karner [83], details are explained below).

Although they are both quite old, many subsequent approaches to testing effort estimation are based on one of them, and some use even both of those methods. We have identified 17 papers describing testing effort estimation methods based on TPA or UCP—there are either propositions of their modification or they are used as parts of some new methods. Each of that two standard methods is used in 10 approaches—TPA is referred to by References [8, 28, 50, 54, 71, 73, 94, 98, 157], while [2, 7, 8, 25, 63, 98, 125, 157, 159, 161] take advantage of UCP. Other analytical approaches are not so common in testing effort estimation literature, and the obvious conclusion is that those two methods must be meaningful for this problem.

The objective of the Test Point Analysis (TPA) [168] is to calculate approximate number of test hours necessary for black-box testing of the system. Veenendaal and Dekkers consider three main factors that influence the number of test hours, i.e., the size of the system measured in function points, the test strategy including the information about tested quality characteristics, and the productivity factor, which is necessary to transform the number of test points into the number of hours. They also indicate that the effort that needs to be spent on white-box testing preceding the black-box testing is counted in the results of Function Point Analysis. Three types of quality testing are regarded in TPA: dynamic implicit tests, dynamic explicit tests, and static tests.

The original Use Case Points (UCP) [83] method was proposed by Karner to estimate resources necessary for the whole development activities. Nageswaran [117] modified it so it can be applied to calculate time of the testing phase. This method mainly assigns weights to use cases and actors depending on their types and considers the complexity factor, arriving at the number of adjusted use case points with the simple formula. Finally, the number of UCP is transformed into man-hours that need to be spent on test planning, writing, and executing. Although the idea of the original UCP and its modification are very similar, the details differ, as the latter approach is suited for test estimation. It led to the inversion of types of actor complexities, change of weights, and combination of technical and environmental complexity factors. The characteristics considered in the technical factors also differ, since Nageswaran uses test-specific features of the project to estimate effort. The final formula of adjusted UCP calculation also is other than in the original version and it rather reminds of the adjusted function points counting formula.

Almeida et al. [7] modify Nageswaran's approach [117] to consider the fact that the usual scenarios of program execution require much more testing that exceptional scenarios. Parvez [125] claims that the original UCP method is based on the features of the project under test, but not on the characteristics of the test team, which in fact influence the amount of testing effort. He extends Use Case Points method with new factors regarding information specific for test cycle length and test team properties [125]. Grover et al. [63] use fuzzy techniques to determine values of parameters for Revised UCP (Re-UCP) method. Sahoo et al. present their own approach for effort estimation on the basis of the UML use case diagram [141] or use case and class diagrams [142], but it covers only effort spent on system tests. Khurana et al. [90] consider only regression testing effort calculation. In their approach, class diagrams are parsed to determine the impact set (i.e., the set of classes influenced by the change in the system) and then the test suite is reduced accordingly [90]. They estimate regression test effort from the reduced test suites (on the basis of the impact set data). Kumar and Beniwal [94] consider application of Test Point Analysis for two types of software projects, in which test stubs and drivers may be used. They set the strict values of TPA parameters and there are only two differences of those values depending on usage of stubs and drivers—all other parameters have the same value, regardless of the testware used. We also have taken advantage of the Test Point Analysis method in our approach to test effort estimation from UML class and sequence diagrams [28]. We have combined TPA with automatic Function Point Analysis to calculate effort in test hours [28] and proposed simple rules to adjust that method to the current versions of standards in software engineering [107, 29, 106].

Some approaches combining analytical TPA or UCP methods with metaheuristics can also be found. Aloka et al. [8] use PSO algorithm to optimize effort calculated with both UCP and TPA methods. Kumari and Kaur [98] combine two metaheuristics. In their approach, first the water wave algorithm is used to optimize parameters for firefly algorithm. Then, the firefly algorithm with such prepared parameters is used to optimize parameters of TPA and UCP. Srivastava et al. also describe approaches to optimize weights of analytical methods of estimation. In Reference [157], they have applied bat algorithm to suit parameters of TPA and UCP; while in Reference [159], cuckoo search algorithm is used to prepare values of UCP parameters.

Machine learning methods are rather rarely used for testing effort estimation. Cotroneo et al. [41] aim at estimating effort of testing of real-time operating system modules using classification and regression approaches. They apply binary classification to divide components into two groups: requiring more and less testing. Moreover, they also use logistic regression to determine the number of lines of code of test programs and the number of test programs itself [41]. Jayakumar and Abran [75] build a group of linear regression models using various variables and compare the results obtained for different methods of Function Points calculation. Kaur and Kaur [85] also build a regression model; however, they aim at estimating testing effort for mobile software. They use COSMIC Function Points and factors specific for mobile applications and testing as variables in their dataset [85]. All solutions mentioned above use metrics as their input. There also exist approaches that utilize artificial neural networks for testing effort estimation. For instance, Abhishek et al. [2] propose a neural network that uses parameters of Nageswaran's UCP method as the input values. Similarly, the neural network created by Felipe et al. [50] requires intermediate values of TPA method as its input. Artificial neural networks for testing effort estimation do not have to obligatorily be combined with analytical methods. Dawson [43] proposes a neural network to calculate effort of unit testing on the basis of the values of three source code metrics, while Mensah et al. [110] combine Mixed-Integer Linear Programming with deep neural networks to select completed software projects that will be used as a dataset for estimation and to determine the testing effort.

There are also some approaches for testing effort estimation based on the Test Point Analysis that seem to be very unreliable. It applies particularly to the publications prepared by Islam et al. [71, 73]. Not only have they copied original TPA description [159] in References [73, 71], but also in the subsequent [72] paper they use TPA improperly and their calculations that aim at demonstrating "their" (i.e., TPA) approach are very unclear in References [73, 105]. The tool described by Garg [54] also belongs to that category. Garg claims that he determined and compared testing effort for projects with and without reused code. In fact, his contribution is limited to setting values of TPA parameters (without any explanation why such values were chosen). Moreover, he determines the parameters values only in one manner, although he claims that the presented results were obtained in two manners. Further, Garg sets the percentage of software project elements to which particular values of TPA parameters can be assigned, similarly to Kumar and Beniwal [94].

Papers based on other approaches can also be untrustworthy. This sad conclusion can be made after reading the articles prepared by Bhatia and Kumar [25] and Sundari [161]. The first alarming observation that we have made is that a quite long fragments of that two papers are identical or very similar. It applies not only to the text, because in both cases the same figure presenting the so-called V-model of software development life cycle is used without indicating the source and even the same formatting (i.e., usage of bolded font) and references to the same publications can be noticed. Moreover, both papers seem to describe estimation technique based on UCP, but the title of Reference [161] indicates that the method presented by Sundari should be based on the Test Case Point Analysis—according to that, we conclude that the title of paper does not match its content, so it could not be regarded as a serious source. In fact, reading these two papers is a very difficult task, as subsequent statements or paragraphs seem to be very weakly related to the previous ones and it is not clear what is the idea behind the described approaches. Finally, after a small investigation, we have understood the reason for all that difficulties. It turned out that those articles are not only suspiciously similar to each other, but they contain copied fragments of two other publications combined in an unclear manner (we believe that there is no reason behind that combination and that it is rather random). Both authors have copied mainly fragments of UCP method description by Clemmons [38], but they have also pasted a part of paper on test execution effort by Aranha and Borba [15] (that publication of those authors was not found during our research, although

we have come across five of their different papers)—moreover, it was done in some strange way, without preserving the original paragraphs or with the deletion of particular words. One difference that was observed between UCP description by Clemmons [38] and its copy by Bhatia and Kumar [25] is that the former paper presents 13 technical complexity factors, while the latter describes 17 such factors. We were not able to detect whether there are more sources that were used to prepare that untrustworthy papers, and we are convinced that they cannot be regarded as good descriptions of testing effort estimation methods. We decided to describe them to show that not all papers emerging during search are equally valuable.

Although TPA and UCP seem to be most common bases for testing effort estimation, we have also observed other well-known approaches underlying some methods of such estimation. We have distinguished three types of such basic approaches, i.e:

- different versions of Function Point Analysis (FPA) method,
- Constructive Cost Model (COCOMO), and
- various methods using the name of Test Case Point Analysis (TCPA).

Bhattacharya et al. [26] combine COCOMO model with Particle Swarm Optimization metaheuristic. Similary to the approaches that use TPA or UCP together with metaheuristics, the objective of the authors is to optimize values of COCOMO parameters by means of PSO algorithm. Further, Srivastava et al. in their papers propose algorithms that combine fuzzy logic with COCOMO [155, 158] or add also fuzzy multiple linear regression to them [156], while Jiang et al. [76] modify the original COCOMO approach (which is defined for effort estimation of the whole software development) so it suits the testing effort estimation problem. However, the latter method is based on completed projects data, and it is not known how it would work in a general case. Ahmad and Samat [5] aim at estimating the cost of quality and testing phase (considered together) by extracting it from the total cost of software project. They calculate cost of the whole project using the combination of two models, but their solutions use also results of Function Point Analysis and a part of COCOMO approach.

We have identified three approaches that claim to have something in common with Test Case Point Analysis method. Nguyen et al. [120] use test case points in their qEstimation process that aims at estimating size and effort of testing phase [120]. Chaudhary and Yadav [36] describe the approach that requires SRS document as an input to calculate test case points. In another paper [37] they compare the results of estimation for test case point method and Function Point Analysis with the conversion factors that allow to transform the function points to person-days. Although they claim to have conducted a comparative analysis of that two approaches, their description looks rather like a simple (but ununderstandable) numerical example of usage of the methods—they do not provide any theoretical background, any formulas that are used in calculation, or any information about the origin of the presented project data. The authors of all of those three papers use the term "test case point" to name the approaches they are adapting, but which are, as far as we are concerned, hardly related to the original Test Case Point Analysis method proposed by Patel et al. [127]. Furthermore, although the first paper refers to the original TCPA method, the other two do not.

Metrics can be regarded as the other kind of approaches to estimation of effort necessary for software testing. For instance, Kashyap et al. [84] propose metrics for testing effort estimation of three-tier object-oriented systems. Other metrics are defined by Kushwaha and Misra [100] or by Periyasamy and Liu [130]. The common characteristic of all mentioned approaches is the fact that values of measures are calculated based on the source code, what implies that they cannot be used early in the project life cycle. The metrics developed by Sharma and Kushwaha [145, 146] are, in turn, calculated from the SRS document. Abhilasha and Sharma [1] and Lazić and Mastorakis [102]

also define measures, or—more precisely—the whole methods used to calculate values of metrics. However, the former method is suited only for regression testing effort calculation, while the latter is meant to determine effort of system testing.

During our research, we have also identified other approaches that can be called analytical, although they do not belong to any of the previous categories. The solution presented by Ghanim [57] requires consideration of the risk of failure of software units. The author convinces that the more critical the part of the system is, the more testing it requires [57]. The approach proposed by Umar [167] utilizes the Cobb-Douglas function to model the effort.

Methods described by Pinkster et al. [131] belong neither to analytical nor to AI solutions category. Those approaches require experience and knowledge of experts on software testing and, as such, cannot be automated.

3.2.1.1 Related work. Comparison of our work with the results presented in other SLRs considering the problem of testing effort estimation leads to very interesting conclusions. We have identified only five reliable and trustworthy such publications, prepared by Adali et al. [3], Araujo and Matieli [17], Jayakumar and Abran [74], and even two research papers by Kaur and Kaur [86, 87]. Researchers indicate the difficulty in finding methods of software testing effort estimation (Reference [17] in general testing estimation context and Reference [86] in the context of the mobile software domain) and the lack of tools for that task [86, 87]. In fact, none of these publications presents the testing effort estimation problem in a broader context—with relation to other research areas—and they do not cover topics connected to testing effort calculation.

Adali et al. [3] and Araujo and Matieli [17], similarly to us, point out the fact that it is very difficult to find publications treating testing effort estimation as a separate problem—there are only a few of them, in comparison to the number of papers on related topics. Testing effort is often estimated as a part of the overall development effort, as can be seen in the results presented by Kaur and Kaur [87]. Furthermore, Araujo and Matieli [17] enumerate the fields related to software testing (but not to testing effort estimation) that they had come across during their research, such as failure prediction, automation, regression testing, fault number estimation, test data, test case generation, and so on [17]. Their observations are very similar to ours—although the main research topic was clearly stated, the majority of found papers do not answer our main questions.

Another interesting observation is that a lot of publications that we have selected in our research are also described in other SLRs. Nineteen out of 39 papers described by Reference [3] (i.e., References [7, 11, 13, 21, 22, 35, 50, 66, 100, 117, 145, 147, 150, 157, 159, 168, 175, 183, 184]; although the reference list presented in Reference [3] contains 44 papers, one of them appears two times because of misunderstanding of foreign names and surnames and 4 others are not referred to in the literature description—that is why we claim their SLR consists of 39 papers), 3 out of 7 publications considered by Reference [17] (i.e., References [76, 120, 183]) and 24 out of 75 articles answering research questions stated in Reference [87] (i.e., References [2, 8, 11, 12, 14, 21, 26, 27, 40, 72, 100, 102, 117, 120, 125, 146, 150, 156, 157, 159, 168, 182, 184]) are the same as considered in our review. However, Kaur and Kaur [86] focus on testing effort estimation for mobile apps and the number of repeating references in comparison to this article is rather small—it is only 3 out of 49 of their identified studies. We have come across 2 of 3 testing effort estimation methods they have mentioned in Reference [86] and the rest of publications they are referring to is either on mobile software development (including effort calculation) or on various aspects of mobile apps, their characteristics, and testing, so in fact many of those papers seem irrelevant to us. Similarly, in Reference [87] they consider the topics of mobile apps development and testing effort together and that is why only one-third of papers they used appear in our results.

Adali et al. in their literature review [3] distinguish three categories of testing effort estimation methods:

(1) based on base software attributes,
(2) based on effort distribution during the whole software development process, and
(3) solutions to resource allocation problem by means of fault prediction or reliability modeling.

The most interesting group of approaches to solve the testing effort estimation task is the first one. The authors indicate that such methods primarily utilize use cases, test cases, or source code as an input. Further, they have found only two methods belonging to the second group (i.e., methods based on the overall project effort) and the papers assigned to the third group aim at proper resource allocation instead of the effort approximation. They have also conducted a survey to examine the current practices in Turkish software industry with regard to methods and tools of testing effort estimation and software quality characteristics important for that process. It turned out that the majority of their respondents use estimation methods based on the overall development effort distribution or Test Case Points, while the rest uses mainly approaches based on metrics, such as LOC, cyclomatic complexity, number of test cases, or function points. Interestingly, the usage of testing effort estimation tools is also very rare: the majority of practitioners either does not use any tool or use MS Office or similar software to support their estimation process [3]. The results of the survey conducted by Adali et al. also indicate that almost all software characteristics they had asked for are considered important or critical for the test estimation by many respondents.

Araujo and Matieli [17] describe eight methods of testing effort estimation. Their results cover the estimation based on the development effort distribution, on the percentage of development time or on historical data, calculations using Function Points, Use Case Points, Test Case Points, or Test Point Analysis, and Simple Estimate Test (SET) method proposed by Matieli (that is, a co-author of the SLR) [17]. The authors of this publication claim that the TPA method is considered as the most consistent among the approaches described in the literature [17].

Kaur and Kaur [87] are focused on the mobile software and its testing effort. They were searching for methods for desktop and mobile applications and for traditional or agile development process. They have identified 25 approaches to testing effort estimation (as a separate task) in traditional development process in 26 papers. Twenty-one out of those 26 papers are also considered in our research. The conclusions from this part of their review are very similar to ours. They indicate that there are practically no tools supporting the estimation process, the UCP and TPA methods are the most commonly used as bases for modifications, and many authors combine other approaches with metaheuristics [87]. We think that some of the tools they have come across are unreliable, which can be seen in Section 3.2.2. Kaur and Kaur also point out the lack of testing effort estimation methods for the mobile software (especially when it is developed according to agile methodology) and the fact that in major number of cases the effort of testing is calculated together with the total development effort.

The same authors have also conducted the SLR accompanied with the survey on testing effort estimation in the context of mobile applications [86]. They aimed at identification of such methods and the characteristics specific to mobile apps that can influence the manner of their testing. They claim that there are a lot of methods for testing effort estimation for traditional software [86], but many types of methods that they mention are based on expert experience or analogy and cannot be automated. In fact, they indicate only 12 examples of traditional testing effort estimation methods and 3 approaches specific for mobile apps. Twelve out of those 15 approaches are described also in this review. During their research they have concluded that in majority of cases, traditional effort estimation techniques are used in mobile software test estimation with some amendments and that

these approaches do not consider characteristics specific for mobile domain. They have identified 14 such characteristics in the literature [86]. Their results indicate the popularity of Function Point Analysis or Test Point Analysis followed by expert-based methods. Findings from their survey also confirm that in most cases traditional estimation methods are adapted to mobile software. Finally, they state that practitioners do not use testing effort estimation tools. Even if some of them do, they usually mention tools not specific to effort estimation, rather to mobile app testing [86]. They observed that there is a need for a tool support and a better knowledge of testing effort estimation methods in the industry. In the context of their research goals, a method suited for mobile software is also necessary.

Jayakumar and Abran [74] analyze existing software testing effort estimation methods in the context of test process outsourcing. They have formulated five criteria of methods evaluation (some of which are claimed to be crucial for outsourced tests), i.e., customer point of view of the requirements, usage of functional size in the method, mathematical validity of approaches and verifiability of their results, and possibility of benchmarking of the estimates. They distinguish five groups of approaches, namely, methods based on expert judgement and rules of thumb, approaches using analogy and work breakdown, techniques built on factors and weights or on size and modern approaches using fuzzy logic or artificial neural networks. They describe a few methods belonging to each group (24 methods in total) and assess each group according to their evaluation criteria. Some of the methods presented in Reference [74], especially among those belonging to the group of modern approaches, are prepared for the whole development process effort estimation and the authors only indicate that they "could be adopted for estimating testing effort as well." Jayakumar and Abran indicate disadvantages of all types of solutions, concluding that estimation methods based on size satisfy the most of their criteria. They have also made interesting remarks on mathematical validity of the methods, including TPA and UCP. In case of both of those methods the authors claim that they violate mathematical rules, ignore units or scales and the methods can only be used as "feel good" approaches, but they are in fact incorrect. In case of the TPA, Jayakumar and Abran do not refer to original publication [168], but to the description prepared by a third party. The explanation of the TPA process provided by them differs from the original one, so the conclusions about the lack of mathematical validity are based on incorrect premises.

To sum up, the general observations from the other SLRs on testing effort estimation problem seem to be quite similar to what is presented in this article. A lot of publications cited by other researchers appear also in our research. None of the other SLRs describes questions related to testing effort estimation that can be pointed out after the database search. There is a significant problem with finding testing effort estimation-specific publications, methods, and tools and with their reliability and correctness. The answers given by practitioners in two independent surveys [3, 86] also indicate either that they do not use any tool for testing effort estimation or that the tools they are using are of more general purpose and are not suited for testing effort estimation.

3.2.2 RQ 2. Which Automatic Tools Are Available for Testing Effort Estimation? The paper prepared by Sahoo and Mohanty [142] seems to be very valuable in context of searching for testing effort estimation tool, but we do not regard it as a tool description, as we do not know whether it has been implemented. That point of view is precisely explained in Section 3.1.1. Except of this one, we could identify only 10 publications describing (already existing) tools related to the problem of testing effort estimation. Unfortunately, they usually are not very useful for the general subject of software testing effort estimation.

In fact, majority of them either addresses a narrow scope of application or seems unreliable. Aranha et al. [16] propose the tool that automates calculation of effort necessary only for test execution, not for the whole testing phase. Tahvili et al. [162] also have developed the tool for

estimation of testing execution time, but in addition to that, they consider only manual test cases execution. Sneed [153] prepared a tool for extracting test cases from different types of software requirements documents. He indicates that, if we can count the number of test conditions, we would also be able to prepare better estimations of testing effort, but the only connection of his software with testing effort estimation is limited to the possibility of counting the number of logical test cases and obtaining testing effort using known productivity of the testers. Bock et al. in References [30, 31] describe the method and tool for estimation of test effort prepared for very specific domain—multi-sensor driver assistance systems. In fact, they calculate error probability of the system and they use it to estimate the amount of effort required to assure the lack of errors [30].

Considering the above explanation, it seems that only five papers describe the tool suited for testing effort estimation in general, but the reality is a bit different. What is interesting, all the tools described in those papers are based on the same test effort estimation method—Test Point Analysis (TPA), introduced by Veenendaal and Dekkers [168]. Unfortunately, they mostly turn out to be of very doubtful quality. Garg [54] hardcodes the values of all TPA parameters and, although he claims that he describes the differences between the results obtained for projects with and without reused code, he does not explain how to set the TPA parameters in both cases. The other three works [71, 73] are written by Islam et al. and seem to us to be of poor quality. In subsequent papers [71, 73] they have placed copied fragments of the original TPA description by Veenendaal and Dekkers, i.e., of Reference [168]. Additionally, their interpretation of the original method is also incorrect. The same can be observed in their description of the test effort estimation tool [72]. They confused software quality characteristics with weights assigned to that characteristics and, as a consequence, their tool calculates so-called quality-dependent factor in a wrong way. It all shows that it is almost impossible to find any trustworthy tool for estimation of effort required for the whole testing phase. To address this issue, we have proposed and implemented the tool [28] based on the TPA method and automatic Function Point Analysis. Its objective is to calculate the number of test hours on the basis of UML class and sequence diagrams and information about the development and testing environment [28, 107]. That tool was recently extended, as can be seen in References [29, 107].

3.2.3 RQ 3. What Are the Measures of Testing Effort? Roman [136] defines the test estimation as a "process of approximation of cost, time or effort that has to be spent for testing." He also indicates that there are two typical measures of effort, namely, cost or time [136]. There also exist other measures derived from the primary ones, product or quotient measures, which are more commonly used [136]. An example of a product measure can be *man-hours*.

The measures of testing effort that we have identified during our survey are presented in Figure 9. The results clearly show that testing effort is measured by the researchers mainly in terms of time. Interestingly, the granularity of time units used in different publications varies a lot. We would expect that the number of months, days, or even hours can be used to measure the effort, but some researchers express the testing effort even in minutes or seconds. We can also observe two approaches for using time as an effort measure. First, the time serves as a primary measure of effort and it can be counted in the number of well-known time units (days, hours, minutes, or even seconds) necessary for testing. Second, there appear "product measures" built on the basis of the time units, such as man-months, man-days, or man-minutes. In both cases, the number of hours (i.e., number of hours or man-hours) seems to be the most popular measure of effort—it is used much more frequently than the less specific units, i.e., days or months. Further, Roman's observation that derived measures are more common than the primary ones [136] can be positively verified on the basis of our results. In the papers related to testing effort estimation

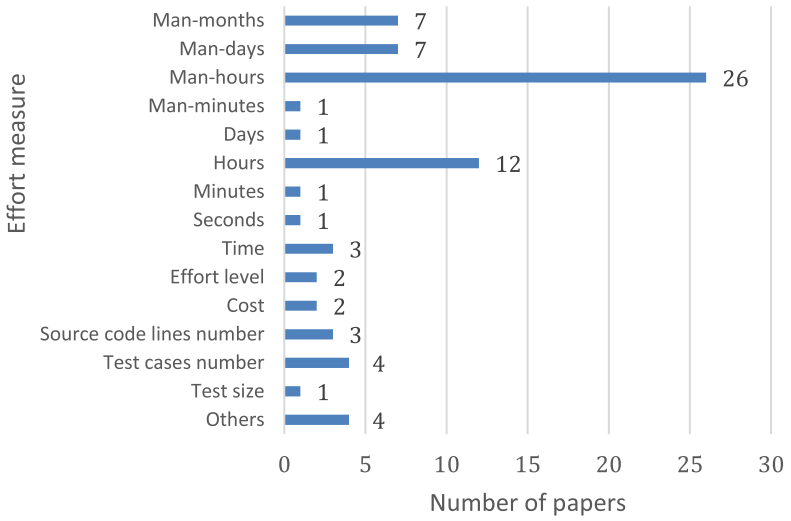

Fig. 9. Distributions of measures of testing effort.

problem, man-days, man-hours, and man-minutes serve as the effort measures in a much greater or equal number of cases, respectively—e.g., effort is measured in man-hours in 26 articles, while hours are used only 12 times. Some authors (Silva et al. [150] and Tahvili et al. [162, 163]) do not specify the unit that they use for time measurement, and they only indicate that they calculate testing effort by means of time.

The other measures of effort used in Roman's definition [136] recalled above appear very rarely in the results of the literature review. Badri, Toure, and Lamontagne [21, 166] use the level of effort for unit testing estimation. Further, the cost is used as a testing effort measure by Ahmad and Samat [5] and Mensah et al. [110].

We also observed that there exist other testing effort measures recognized by the researchers and related directly to the software artifacts. Ahmad and Samat [5] use the number of source code instructions to calculate an effort, while Badri et al. [19, 20] count testing effort in terms of the number of test lines of code. Further, Gupta and Kushwaha [65], Martins and Melo [109], and Zhu et al. [183, 184] express testing effort in the number of test cases. Test size is used as such measure by Jiang et al. [76]. Some exceptional effort measures can also be found in the literature. Kazerouni et al. [88] treat testing effort as "the percentage of effort that was devoted to writing test code." However, Bock et al. [30, 32] use error probabilities to determine such amount of testing effort (testing length [32]) that after the test execution no errors should occur in the sensors system [30]. Their specific definition and measure of testing effort is probably caused by the fact that they consider the automotive systems domain.

3.2.4 RQ 4. Which Research Topics Are Related to Testing Effort Estimation? In Section 3.1.2 we have identified 13 main objectives of papers selected for this literature review. They all have been found during the search for the testing effort estimation method and tools, as explained in Section 2. Consequently, we regard all the other aims of obtained articles as related to the question of software testing effort estimation.

It turns out that research topics related to the testing effort estimation problem usually belong to the domain of software testing. There is one exception to that rule—the problem of **software development effort estimation** (i.e., estimation of effort for the whole testing process, not only for the testing phase). We can also observe two major problems related to testing effort

estimation; that is, **testing effort reduction** (without determination of the amount of such effort) and **resource allocation**. Further, there are approaches for **software reliability growth modeling** and many various attempts to testing process improvement: **automatic test generation** or **other test automation** (including, for instance, automatic test recommendation), **test prioritization, reduction,** and **selection**. Finally, the topics related to fault-proneness prediction seem to be closely connected to testing effort estimation. That includes the mentioned **fault-proneness prediction, fault number prediction**, and **fault-prone modules ranking**.

As we explained in Section 3.1.2, approaches used for testing phase improvement (i.e., automatic test generation, test prioritization, reduction, and selection) also have an objective to minimize testing effort. Similarly, all attempts to software reliability growth modeling and fault-proneness prediction can be regarded as solution for testing resource allocation problem, too.

It should also be noticed that in the domain of testing effort estimation there are different subgroups of solutions for various versions of that problem. Particularly, it appears that there are a lot of approaches for test execution effort estimation (that is, only the effort required for test execution is estimated). Detailed descriptions and examples of all research topics identified during the survey and presented here can be found in Section 3.1.2.

4 THREATS TO VALIDITY

We are aware that our survey, like others, has some limitations. The main source of the threats to validity of this study is the manner of conducting the literature review. The number of searched data sources is by nature limited, and although we also conducted the research on the Internet, there is still a chance that we have missed some important research. Moreover, in majority of papers only abstracts and conclusions were analyzed. That is the result of the fact that a huge amount of publications did not cover testing effort estimation problem, but are dedicated to other topics. The other danger to the correctness of our observations is that the study was performed in three parts, in different periods of time. Furthermore, every time different papers were found and although the publication years were limited during the database search, there was no such limit in the Internet search, so each time we found few papers published not as recently as we had expected. If we regarded them vital, we included them in our results. However, we have come across some papers new to us, but published before the beginning of our research, when the whole SLR was already conducted and described. Such publications, e.g., References [33, 62], are omitted in this article, as their addition would require the changes in presented numerical data. Although we have prepared some quantitative data in the form of number of papers belonging to each category of each classification type, we have not conducted any numerical quality assessment of testing effort estimation methods. We have focused only on the qualitative description of characteristics of the identified approaches. The number of papers identified during each phase of our research also varies. There is a disproportion in number of papers from different periods of search: There are many papers from the short periods of time, i.e., 2016–2018 and 2018–2019. We have also excluded some articles on topics other than testing effort estimation in the second stage of the research to make our work more focused on our main goal. Finally, this SLR was prepared without any special tool, nor any environment, which would significantly facilitate the whole process. The only facilitations we have used were the search engines and MS Excel, but the whole data extraction, bibliography management, and consistency assurance was done manually.

All the aforementioned aspects may have some impact on the results described in this article. It is possible that we have missed some important research and obviously we were not able to include all the interesting papers in our survey. It results from our research decisions (e.g., the sources searched, the exclusion criteria, the selected period of time, and the conduction of research in three stages) and, of course, such risk cannot be avoided. We could have also drawn some incorrect

conclusions about some papers on topics related to testing effort estimation, as in their case mainly the abstracts were analyzed. It may cause some numerical values about the number of papers belonging to categories of classification types to not be thorough. However, the sum of papers belonging to each category is not equal to the total number of papers, as categories were assigned independently of each other (some papers are assigned to many categories of a classification type, some are given none). For some readers it can be confusing, but we believe it was the only way to classify papers broadly according to all criteria on the basis of the available information. Some mistakes can be also introduced by the manual management of all data (e.g., references). Finally, it could be difficult to repeat the survey with identical results, as some of the decisions we made were either informal or subjective, we also performed it in different periods of time. However, all those drawbacks and possible threats do not change the fact that the general observations presented here are undeniable. Testing effort estimation-related literature is certainly overwhelmed by the literature on related topics. There are many difficulties in finding reliable and useful methods of estimation, and there is almost no tool support. Last, but not least, there are many interesting problems related to testing effort estimation, and the classification of them in many ways would not change significantly even if the systematic literature review was performed a bit differently.

5 CONCLUSIONS

Despite the fact that software testing effort estimation seems to be a very important task, there exist extremely limited number of scientific approaches to solve that problem. In this article, we have presented the current state-of-the-art of the testing estimation domain, and we have discussed many related topics. The undeniable contribution of this work is that it not only enumerates tasks related to testing effort calculation, but it also comprehensively describes them and illustrates the problems in finding specific solutions prepared for test estimation. To the best of our knowledge, this is the first work that treats the problem of testing effort estimation in the context of all related software-engineering topics, and we believe it is our main contribution. Moreover, some solutions that claim to solve the testing estimation problem turn out to be of very doubtful quality, and here we have identified and criticized them. We have classified all selected papers according to seven criteria based on our research questions. The papers covering testing effort estimation question directly form less than a half of all selected papers and only a small part of all search results. Those publications were carefully analyzed and described. In fact, the other advantage of this SLR is that it proves the lack of testing effort-specific solutions using the concrete numbers—the number of papers on testing effort estimation in contrast to the (already filtered) number of papers on other topics. Although we made our best efforts to perform this survey in a systematic and clear manner, it is not free from some threats to validity, e.g., the limited number of analyzed papers and data sources, possibility of manual errors introduction, some subjective or informal decisions that cause the difficulty in exact repetition of this SLR. They may have a small impact on the numeric results presented in this article but do not affect the general conclusions and observations.

In this SLR, we explained that it is difficult to find any solutions for the testing effort estimation problem among many papers on related topics. There is lack of analytical methods and often experience-based approaches are used. There are almost no reliable tools for testing effort estimation, and the industry also does not use any testing effort estimation-specific tool support. The most popular analytical methods among the academic researchers are TPA [168] and UCP [117]. These solutions are about 20 years old and although they are still modified by different new approaches, they lack modern methods. The growing interest in the area of test estimation can be observed in the past 10 years, but still there are almost no interesting results. It all indicates that testing effort estimation considered as a separate problem should be further investigated.

REFERENCES

[1] Sharma A. Abhilasha. 2013. Test effort estimation in regression testing. In *Proceedings of the IEEE International Conference in MOOC Innovation and Technology in Education*. 343–348. DOI: https://doi.org/10.1109/MITE.2013.6756364

[2] C. Abhishek, V. P. Kumar, H. Vitta, and P. R. Srivastava. 2010. Test effort estimation using neural network. *J. Softw. Eng. App.* 3, 4 (2010). DOI: https://doi.org/10.4236/jsea.2010.34038

[3] O. E. Adali, N. A. Karagöz, Z. Gürel, T. Tahir, and C. Gencel. 2017. Software test effort estimation: state of the art in Turkish software industry. In *Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications*. 412–420, DOI: https://doi.org/10.1109/SEAA.2017.72

[4] W. Afzal, A. N. Ghazi, J. Itkonen, R. Torkar, A. Andrews, and K. Bhatti. 2015. An experiment on the effectiveness and efficiency of exploratory testing. *Empir. Softw. Eng.* 20, 3 (2015), 844–878. Springer Science + Business Media. DOI: https://doi.org/10.1007/s10664-014-9301-4

[5] S. F. Ahmad and P. A. Samat. 2018. Extraction cost of quality and testing in software project. In *Proceedings of the IEEE Conference on E-Learning, E-Management and E-Services (IC3e'18)*. 109–115. DOI: https://doi.org/10.1109/IC3e.2018.8632624

[6] M. Ahmed and R. Ibrahim. 2014. Improving effectiveness of testing using reusability factor. In *Proceedings of the International Conference on Computer and Information Science*. 1–5. DOI: https://doi.org/10.1109/ICCOINS.2014.6868451

[7] É. R. C. de Almeida, B. T. de Abreu, and R. Moraes. 2009. An alternative approach to test effort estimation based on use cases. In *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops*. IEEE. 279–288. DOI: https://doi.org/10.1109/ICST.2009.31

[8] S. Aloka, P. Singh, G. Rakshit, and P. R. Srivastava. 2011. Test effort estimation-particle swarm optimization based approach. In *Contemporary Computing*, S. Aluru et al. (Eds.). Springer-Verlag Berlin. 463–474. DOI: https://doi.org/10.1007/978-3-642-22606-9_46

[9] H. Aman, T. Nakano, H. Ogasawara, and M. Kawahara. 2017. A test case recommendation method based on morphological analysis, clustering and the Mahalanobis-Taguchi method. In *Proceedings of the 10th IEEE International Conference on Software Testing, Verification and Validation Workshops*. 29–35. DOI: https://doi.org/10.1109/ICSTW.2017.9

[10] A. Ansari, M. B. Shagufta, A. Sadaf Fatima, and S. Tehreem. 2017. Constructing test cases using natural language processing. In *Proceedings of the 3rd IEEE International Conference on Advanced in Electrical, Electronics, Information, Communication and Bio-Informatics*, P. L. N. Ramesh et al. (Eds.). 95–99. DOI: https://doi.org/10.1109/AEEICB.2017.7972390

[11] E. Aranha and P. Borba. 2007. An estimation model for test execution effort. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement*. 107–116. DOI: https://doi.org/10.1109/ESEM.2007.73

[12] E. Aranha and P. Borba. 2007. Empirical studies of test execution effort estimation based on test characteristics and risk factors. *Proceedings of the 2nd International Doctoral Symposium on Empirical Software Engineering (IDoESE'07)*. Retrieved from https://pdfs.semanticscholar.org/2fa8/bc98114c0e32d2e59c1829e1af2b6f20fc7c.pdf.

[13] E. Aranha and P. Borba. 2009. Estimating manual test execution effort and capacity based on execution points. *Int. J. Comp. App.* 31, 3 (2009), 167–172. *ACTA Press*. DOI: 10.1080/1206212X.2009.11441938

[14] E. Aranha and P. Borba. 2007. Test effort estimation models based on test specifications. In *Proceedings of the Conference on Testing Academic and Industrial Practice and Research Techniques*. 67–71. DOI: 10.1109/TAIC.PART.2007.29

[15] E. Aranha and P. Borba. 2019. Test execution effort and capacity estimation. Retrieved from https://pdfs.semanticscholar.org/be9f/b7e4ffd6e9ea536b500f8697558e708a7212.pdf.

[16] E. Aranha, F. de Almeida, T. Diniz, V. Fontes, and P. Borba. 2008. Automated test execution effort estimation based on functional test specifications. Retrieved from https://www.semanticscholar.org/paper/Automated-Test-Execution-Effort-Estimation-Based-on-Aranha-Almeida/a7db957392c3e07a498a1556383abd7822a02c6f.

[17] F. O. de Araujo and L. V. Matieli. 2017. Software testing estimation: Bibliographic survey in Brazilian and international environments. *Braz. J.Op. Prod. Manag.* 14, 1 (2017), 10–18. DOI: 10.14488/BJOPM.2017.v14.n1.a2

[18] M. D. Ashraf and N. U. Janjua. 2011. Test execution effort estimation (TEEE) model in extreme programming. *Int. J. Rev. Comp.* 8 (2011), 35–45. Little Lion Scientific.

[19] M. Badri, L. Badri, and W. Flageol. 2013. On the relationship between use cases and test suites size: An exploratory study. *ACM SIGSOFT Softw. Eng. Notes* 38, 4 (2013), 1–5. DOI: 10.1145/2492248.2492261

[20] M. Badri, L. Badri, W. Flageol, and F. Toure. 2017. Investigating the accuracy of test code size prediction using use case metrics and machine learning algorithms: An empirical study. In *Proceedings of the International Conference on Machine Learning and Software Computing*. 25–33. DOI: 10.1145/3036290.¿3036323

[21] M. Badri, F. Toure, and L. Lamontagne. 2015. Predicting unit testing effort levels of classes: An exploratory study based on multinomial logistic regression modeling. *Procedia Comput. Sci.* 62 (2015), 529–538. Elsevier. DOI: 10.1016/j.procs.2015.08.528

[22] K. Bareja and A. Singhal. 2015. A review of estimation techniques to reduce testing efforts in software development. In *Proceedings of the 5th International Conference on Advanced Computing & Communication Technologies*. 541–546. DOI: 10.1109/ACCT.2015.110

[23] T. R. Benala and R. Mall. 2018. DABE: Differential evolution in analogy-based software development effort estimation. *Swarm Evol. Comput.* 38 (2018), 158–172. Elsevier. DOI: 10.1016/j.swevo.2017.07.009

[24] A. Bertolino, R. Mirandola, and E. Peciola. 1996. A case study in branch testing automation. In *Achieving Quality in Software*, S. Bologna et al. (Eds.). Springer Science+Business Media, 369–380. DOI: 10.1007/978-0-387-34869-8_30

[25] P. K. Bhatia and G. Kumar. 2011. Role of technical complexity factors in test effort estimation using use case points. *Int. J. Softw. Eng. Res. Pract.* 1, 3 (2011), 5–12.

[26] P. Bhattacharya, P. R. Srivastava, and B. Prasad. 2012. Software test effort estimation using particle swarm optimization. In *Proceedings of the International Conference on Information Systems Design and Intelligent Applications*. 827–835. DOI: 10.1007/978-3-642-27443-5_95

[27] A. Bhattacharyya and T. Malgazhdarov. 2016. PredSym: Estimating software testing budget for a bug-free release. In *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation*. 16–22. DOI: 10.1145/2994291.2994294

[28] I. Bluemke and A. Malanowska. 2020. Tool for assessment of testing effort. In *Proceedings of the 14th International Conference on Dependability of Computer Systems*. 69–79. DOI 10.1007/978-3-030-19501-4_7

[29] I. Bluemke and A. Malanowska. 2020. Usage of UML combined fragments in automatic function point analysis. In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering – Vol. 1: ENASE*. 305–312. DOI: 10.5220/0009348303050312

[30] F. Bock, S. Siegl, P. Bazan, P. Buchholz, and R. German. 2018. Reliability and test effort analysis of multi-sensor driver assistance systems. *J. Syst. Archit.* 85–86, 1-13. Elsevier. DOI: 10.1016/j.sysarc.2018.01.006

[31] F. Bock, S. Siegl, and R. German. 2017. Analytical test effort estimation for multisensor driver assistance systems. In *Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'17)*. 239–246. DOI: 10.1109/SEAA.2017.49

[32] F. Bock, S. Siegl, and R. German. 2016. Mathematical test effort estimation for dependability assessment of sensor-based driver assistance systems. In *Proceedings of the 42nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'16)*. 222–226. DOI: 10.1109/SEAA.2016.49

[33] E. Borandag, F. Yucalar, and S. Z. Erdogan. 2016. A case study for the software size estimation through MK II FPA and FP methods. *Int. J. Comp. App. Technol.* 53. 4 (2016), 309–314.

[34] M. Böhme. 2018. STADS: Software testing as species discovery. *ACM Trans. Softw. Eng. Methodol.* 27. 2 (2018). DOI: 10.1145/3210309

[35] F. Calzolari, P. Tonella, and G. Antoniol. 2001. Maintenance and testing effort modeled by linear and nonlinear dynamic systems. *Inf. Softw. Technol.* 43, 8 (2001), 477–486. DOI: 10.1016/S0950-5849(01)00156-2

[36] P. Chaudhary and C. S. Yadav. 2012. An approach for calculating the effort needed on testing projects. *Int. J. Adv. Res. Comput. Eng. Technol.* 1. 1 (2012), 35–40.

[37] P. Chaudhary and C. S. Yadav. 2012. Optimizing test effort estimation-a comparative analysis. *Int. J. Sci. Res. Eng. Technol.* 1. 2 (2012), 18–20.

[38] R. K. Clemmons. 2006. Project estimation with use case points. *CrossTalk: J. Defense Softw. Eng.* 19, 2 (2006), 18–22. Retrieved from http://www.royclemmons.com/articles/docs/0602Clemmons.pdf.

[39] International Journal of Engineering Research and Applications (IJERA). 2015. In *National Conference on Developments, Advances and Trends in Engineering Sciences (NCDATES'15)*. CMR Engineering College, 12–19.

[40] G. de Cleva Farto and A. T. Endo. 2017. Reuse of model-based tests in mobile apps. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*. 184–193. DOI: 10.1145/3131151.3131160

[41] D. Cotroneo, D. D. Leo, R. Natella, and R. Pietrantuono. 2016. Prediction of the testing effort for the safety certification of open-source software: A case study on a real-time operating system. In *Proceedings of the 12th European Dependable Computer Conference*. 141–152. DOI: 10.1109/EDCC.2016.22

[42] G. Czibula, I. G. Czibula, and Z. Marian. 2018. An effective approach for determining the class integration test order using reinforcement learning. *Appl. Soft Comput.* 65 (2018), 517–530. Elsevier. DOI: 10.1016/j.asoc.2018.01.042

[43] C. W. Dawson. 1998. An artificial neural network approach to software testing effort estimation. *Trans. Inf. Commun. Technol.* 20. WIT Press. DOI: 10.2495/AI980361

[44] V. Debroy and W. E. Wong. 2011. On the estimation of adequate test set size using fault failure rates. *J. Syst. Softw.* 84, 4 (2011), 587–602. Elsevier. DOI: 10.1016/j.jss.2010.07.025

[45] R. C. G. Dhanajayan and S. A. Pillai. 2017. SLMBC: Spiral life cycle model-based Bayesian classification technique for efficient software fault prediction and classification. *Soft Comput.* 21, 2 (2017), 403–415. DOI: 10.1007/s00500-016-2316-6

[46] M. Elallaoui, K. Nafil, R. Touahni, and R. Messoussi. 2016. Automated model driven testing using AndroMDA and UML2 testing profile in scrum process. *Procedia Comput. Sci.* 83 (2016), 221–228. Elsevier. DOI: 10.1016/j.procs.2016.04.119

[47] F. Elberzhager, A. Rosbach, J. Münch, and R. Eschbach. 2012. Reducing test effort: A systematic mapping study on existing approaches. *Inf. Softw. Technol.* 54, 10 (2012), 1092–1106. Elsevier. DOI: 10.1016/j.infsof.2012.04.007

[48] R. Elghondakly, S. Moussa, and N. Badr. 2016. A comprehensive study for software testing and test cases generation paradigms. In *Proceedings of the International Conference on Internet of Things and Cloud Computing*, ACM. DOI: 10.1145/2896387.2896435

[49] A. T. Endo and A. Simao. 2013. Evaluating test suite characteristics, cost, and effectiveness of FSM-based testing methods. *Inf. Softw. Technol.* 55, 6 (2013), 1045–1062. Elsevier. DOI: 10.1016/j.infsof.2013.01.001

[50] N. F. Felipe, R. P. Cavalcanti, E. H. B. Maia, W. P. Amaral, A. C. Farnese, L. D. Tavares, E. S. J. de Faria, C. I. P. da Silva e Pádua, and W. de Pádua Paula Filho. 2014. A comparative study of three test effort estimation methods. *Rev. Cubana Cienc. Informát.* 8, Especial Uciencia (2014), 1–13. Universidad de las Ciencias Informáticas.

[51] J. Ferrer, F. Chicano, and E. Alba. 2013. Estimating software testing complexity. *Inf. Softw. Technol.* 55, 12 (2013), 2125–2139. Elsevier. DOI: 10.1016/j.infsof.2013.07.007

[52] L. Fiondella and S. S. Gokhale. 2012. Optimal allocation of testing effort considering software architecture. *IEEE Trans. Reliab.* 61, 2 (2012), 580–589. DOI: 10.1109/TR.2012.2192016

[53] D. Flemström, P. Potena, D. Sundmark, W. Afzal, and M. Bohlin. 2018. Similarity-based prioritization of test case automation. *Softw. Qual. J.* Springer US. DOI: 10.1007/s11219-017-9401-7

[54] P. Garg. 2011. Simulator to calculate test efforts by using reusability of code. *Int. J. Adv. Res. Comput. Sci.* 2, 4 (2011) 273–276. Genxcellence Publications. DOI: 10.26483/ijarcs.v2i4.622

[55] V. Garousi and M. V. Mäntylä. 2016. A systematic literature review of literature reviews in software testing. *Inf. Softw. Technol.* 80 (2016), 195–216. Elsevier. DOI: 10.1016/j.infsof.2016.09.002

[56] G. Gay, A. Rajan, M. Staats, M. Whalen, and M. P. E. Heimdahl. 2016. The effect of program and model structure on the effectiveness of MC/DC test adequacy coverage. *ACM Trans. Softw. Eng. Methodol.* 25, 3 (2016), 25. DOI: 10.1145/2934672

[57] Y. Ghanim. 2017. Risk-based test estimation. In *Proceedings of the 3rd Africa and Middle East Conference on Software Engineering*. 37–42. DOI: 10.1145/3178298.3178302

[58] A. S. Ghiduk, M. R. Girgis, and M. H. Shehata. 2017. Higher order mutation testing: A systematic literature review. *Comput. Sci. Rev.* 25 (2017), 29–48. Elsevier. DOI: 10.1016/j.cosrev.2017.06.001

[59] R. Göldner. 2001. A cost-benefit model for software testing. In *Software Quality: State of the Art in Management, Testing, and Tools*. M. Wieczorek et al. (Eds.). 126–134. DOI: 10.1007/978-3-642-56529-8_9

[60] G. Grano, F. Palomba, and H. C. Gall. 2019. Lightweight assessment of test-case effectiveness using source-code-quality indicators. *IEEE Trans. Softw. Eng.* DOI: 10.1109/TSE.2019.2903057.

[61] A. Groce, J. Holmes, and K. Kellar. 2017. One test to rule them all. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 1–11. DOI: 10.1145/3092703.3092704

[62] M. Grover and P. K. Bhatia. 2016. Analytical review of software testing effort estimation using case point method. In *Proceedings of the 6th International Conference on Advanced Computing & Communication Technologies (ACCT'16)*. 451–455. DOI: 10.3850/978-981-11-0783-2 451

[63] M. Grover, P. K. Bhatia, and H. Mittal. 2017. Estimating software test effort based on revised UCP model using fuzzy technique. In *Proceedings of the Conference on Information and Communication Technologies for Intelligent Systems (ICTIS'17)*, S. C. Satapathy et al. (Eds.). 490–498. DOI: 10.1007/978-3-319-63673-3_59

[64] A. Gupta and P. Jalote. 2008. An approach for experimentally evaluating effectiveness and efficiency of coverage criteria for software testing. *Int. J. Softw. Tools Technol. Transf.* 10, 2 (2008), 145–160. DOI: 10.1007/s10009-007-0059-5

[65] A. Gupta and D. S. Kushwaha. 2016. Use case-based software change analysis and reducing regression test effort. In *Proceedings of the International Congress on Information and Communication Technology (ICICT'15)*. 459–466. DOI: 10.1007/978-981-10-0767-5_48

[66] R. Harrison and L. G. Samaraweera. 1996. Using test case metrics to predict code quality and effort. *ACM SIGSOFT Softw. Eng. Notes* 21, 5 (1996), 78–88. DOI: 10.1145/235969.235993

[67] M. M. Hassan, W. Afzal, B. Lindström, S. M. A. Shah, S. F. Andler, and M. Blom. 2016. Testability and software performance: A systematic mapping study. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 1566–1569. DOI: 10.1145/2851613.2851978

[68] N. E. Holt, L. C. Briand, and R. Torkar. 2014. Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study. *Inf. Softw. Technol.* 56, 8 (2014), 890–910. Elsevier. DOI: 10.1016/j.infsof.2014.02.011

[69] C. Y. Huang, S. Y. Kuo, and M. R. Lyu. 2007. An assessment of testing-effort dependent software reliability growth models. *IEEE Trans. Reliab.* 56, 2 (2007), 198–211. DOI: 10.1109/TR.2007.895301

[70] A. Idri, M. Hosni, and A. Abran. 2016. Improved estimation of software development effort using classical and fuzzy analogy ensembles. *Appl. Soft Comput.* 49 (2016), 990–1019. Elsevier. DOI: 10.1016/j.asoc.2016.08.012

[71] S. Islam, B. B. Pathik, M. H. Khan, and M. M. Habib. 2014. A novel tool for reducing time and cost at software test estimation: A use cases and functions based approach. In *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*. 312–316. DOI: 10.1109/IEEM.2014.7058650

[72] S. Islam, B. B. Pathik, M. H. Khan, and M. Habib. 2016. Software test estimation tool: Comparable with COCOMOII model. In *Proceedings of the International Conference on Industrial Engineering and Engineering Management*. 204–208. DOI: 10.1109/IEEM.2016.7797865

[73] S. Islam, B. B. Pathik, M. H. Khan, and M. Habib. 2013. Software test estimation tools using use cases and functions. In *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*. 390–394. DOI: 10.1109/IEEM.2013.6962440

[74] K. R. Jayakumar and A. Abran. 2013. A survey of software test estimation techniques. *J. Softw. Eng. Appl.* 6, 10A (2013), 47–52. DOI: 10.4236/jsea.2013.610A006

[75] K. R. Jayakumar and A. Abran. 2017. Estimation models for software functional test effort. *J. Softw. Eng. Appl.* 10, 4 (2017), 338–353. DOI: 10.4236/jsea.2017.104020

[76] L. X. Jiang, W. J. Han, C. C. Yan, and B. Y. Shi. 2012. Research on size estimation model for software system test based on testing steps and its application. In *Proceedings of the International Conference on Computer Science and Information Processing*. 1245–1248. DOI: 10.1109/CSIP.2012.6309085

[77] P. Jodpimai, P. Sophatsathit, and C. Lursinsap. 2018. Re-estimating software effort using prior phase efforts and data mining techniques. *Innov. Syst. Softw. Eng.* 14, 3 (2018), 209–228. DOI: 10.1007/s11334-018-0311-z

[78] P. Johri, M. Nasar, S. Das, and M. Kumar. 2016. Open source software reliability growth models for distributed environment based on component-specific testing-efforts. In *Proceedings of the 2nd International Conference on Information and Communication Technology for Competitive Strategies*. 75. DOI: 10.1145/2905055.2905283

[79] L. P. Kafle. 2014. An empirical study on software test effort estimation. *Int. J. Softw. Comput. Artif. Intell.* 2, 2 (2014), 96–106. Institute of Research and Journals.

[80] S. Kappler. 2016. Finding and breaking test dependencies to speed up test execution. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 1136–1138. DOI: 10.1145/2950290.2983974

[81] P. K. Kapur, P. S. Grover, and S. Younes. 1994. Modelling an imperfect debugging phenomenon with testing effort. In *Proceedings of the 5th International Symposium on Software Reliability Engineering*. 178–183. DOI: 10.1109/ISSRE.1994.341371

[82] P. K. Kapur, G. Mishra, and A. K. Shrivastava. 2016. A generalized framework for modelling multi up-gradations of a software with testing effort and change point. In *Proceedings of the International Conference on Innovation and Challenges in Cyber Security*. 129–134. DOI: 10.1109/ICICCS.2016.7542348

[83] G. Karner. 1993. Resource estimation for objectory projects. *Objective Systems SF AB*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.604.7842&rep=rep1&type=pdf.

[84] N. Kashyap, P. Vats, and M. Mandot. 2017. AVINASH—A three tier architectural metric suit for the effort estimation in testing of OOS. In *Proceedings of the International Conference on Intelligent Communication and Computer Techniques (ICCT'17)*. 36–41. DOI: 10.1109/INTELCCT.2017.8324017

[85] A. Kaur and K. Kaur. 2019. A COSMIC function points based test effort estimation model for mobile applications. *J. King Saud Univ. Comput. Inf. Sci.* Elsevier. DOI: 10.1016/j.jksuci.2019.03.001

[86] A. Kaur and K. Kaur. 2019. Investigation on test effort estimation of mobile applications: Systematic literature review and survey. *Inf. Softw. Technol.* 110 (2019), 56–77. Elsevier. DOI: 10.1016/j.infsof.2019.02.003

[87] A. Kaur and K. Kaur. 2018. Systematic literature review of mobile application development and testing effort estimation. *J. King Saud Univ. Comput. Inf. Sci.* Elsevier. DOI: 10.1016/j.jksuci.2018.11.002

[88] A. M. Kazerouni, C. A. Shaffer, S. H. Edwards, and F. Servant. 2019. Assessing incremental testing practices and their impact on project outcomes. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, 407–413. DOI: 10.1145/3287324.3287366

[89] S. K. Khatri, D. Kumar, A. Dwivedi, and N. Mrinal. 2012. Software reliability growth model with testing effort using learning function. In *Proceedings of the 6th International Conference on Software Engineering*. 1–5. DOI: 10.1109/CONSEG.2012.6349470

[90] P. Khurana, A. Tripathi, and D. S. Kushwaha. 2013. Change impact analysis and its regression test effort estimation. In *Proceedings of the 3rd IEEE International Advance Computer Conference (IACC'13)*. 1420–1424. DOI: 10.1109/IAdCC.2013.6514435

[91] B. Kitchenham. 2004. *Procedures for Performing Systematic Reviews*. Joint Technical Report, Software Engineering Group, Dept. of Computer Science, Keele University. Retrieved from http://www.inf.ufsc.br/~aldo.vw/kitchenham.pdf.

[92] Y. Koroglu, A. Sen, D. Kutluay, A. Bayraktar, Y. Tosun, M. Cinar, and H. Kaya. 2016. Defect prediction on a legacy industrial software: A case study on software with few defects. In *Proceedings of the 4th International Workshop on Conducting Empirical Studies in Industry*. ACM, 14–20. DOI: 10.1145/2896839.2896843

[93] F. Křikava and J. Vitek. 2018. Tests from traces: Automated unit test extraction for R. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 232–241. DOI: 10.1145/3213846.3213863

[94] A. Kumar and V. Beniwal. 2012. Test effort estimation with & without stub and driver using test point analysis (TPA). *Int. J. Eng. Res. Technol.* 1, 7 (2012). ISSN: 2278–0181.

[95] G. Kumar and P. K. Bhatia. 2013. Software testing optimization through test suite reduction using fuzzy clustering. *CSI Trans. ICT* 1, 3 (2013), 253–260. DOI: 10.1007/s40012-013-0023-3

[96] L. Kumar and A. Sureka. 2018. Feature selection techniques to counter class imbalance problem for aging related bug prediction: Aging related bug prediction. In *Proceedings of the 11th Innovations in Software Engineering Conference*. ACM. 2. DOI: 10.1145/3172871.3172872

[97] P. Kumari, N. Bakshi, and Y. Pathania. 2015. Test effort estimation and its techniques. *Int. J. Technol. Res. Eng.* 2, 8 (2015), 1514–1516.

[98] P. Kumari and G. Kaur. 2015. A hybrid firefly-water wave algorithm for effort estimation of software testing. *Int. J. Comput. Sci. Netw.* 4. 4 (2015), 625–631. ISSN: 2277-5420.

[99] S. Y. Kuo, C. Y. Huang, and M. R. Lyu. 2001. Framework for modeling software reliability, using various testing-efforts and fault-detection rates. *IEEE Trans. Reliab.* 50, 3 (2001), 310–320. DOI: 10.1109/24.974129

[100] D. S. Kushwaha and A. K. Misra. 2008. Software test effort estimation. *ACM SIGSOFT Softw. Eng. Notes* 33, 3 (2008), 6. DOI: 10.1145/1360602.1361211

[101] R. Lachmann, M. Felderer, M. Nieke, S. Schulze, C. Seidl, and I. Schaefer. 2017. Multi-objective black-box test case selection for system testing. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 1311–1318. DOI: 10.1145/3071178.3071189

[102] L. Lazić and N. Mastorakis. 2009. The COTECOMO: COnstractive test effort COst model. In *Proceedings of the European Computer Conference*. 89–110. DOI: 10.1007/978-0-387-85437-3_9

[103] X. Li, M. Xie, and S. H. Ng. 2010. Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points. *Appl. Math. Model.* 34, 11 (2010), 3560–3570. Elsevier. DOI: 10.1016/j.apm.2010.03.006

[104] J. H. Lo. 2005. An algorithm to allocate the testing-effort expenditures based on sensitive analysis method for software module systems. In *Proceedings of the IEEE Region 10 Conference (TENCON'05)*. 1–6. DOI: 10.1109/TENCON.2005.301151

[105] A. Malanowska. 2017. *Testing Effort Assessment*. BSc thesis. Warsaw University of Technology, Institute of Computer Science (in Polish).

[106] A. Malanowska. 2019. *Improving Testing Effort Estimation Method with UML Combined Fragments and ISO/IEC 25010:2011 Software Quality Model Support*. MSc thesis. Warsaw University of Technology, Institute of Computer Science (in Polish).

[107] A. Malanowska and I. Bluemke. 2020. ISO 25010 support in test point analysis for testing effort estimation. In *Integrating Research and Practice in Software Engineering*, S. Jarzabek et al. (Eds.). 209–222. DOI: 10.1007/978-3-030-26574-8_15

[108] M. Marré and A. Bertolino. 1996. Reducing and estimating the cost of test coverage criteria. In *Proceedings of the 18th International Conference on Software Engineering*. 486–494. ISBN: 0-8186-7246-3.

[109] A. L. Martins and A. C. V. de Melo. 2016. Can you certify your software to MC/DC?: A static analysis approach to account for the number test cases. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. ACM. DOI: 10.1145/2993288.2993290

[110] S. Mensah, J. Keung, K. E. Bennin, and M. F. Bosu. 2016. Multi-objective optimization for software testing effort estimation. In *Proceedings of the 28th International Conference on Software Engineering & Knowledge Engineering*. 527–530. DOI: 10.18293/SEKE2016-017

[111] S. Mensah, J. Keung, M. F. Bosu, and K. E. Bennin. 2018. Duplex output software effort estimation model with self-guided interpretation. *Inf. Softw. Technol.* 94 (2018), 1–13. Elsevier. DOI: 10.1016/j.infsof.2017.09.010

[112] D. P. P. Mesquita, L. S. Rocha, J. P. P. Gomes, and A. R. R. Neto. 2016. Classification with reject option for software defect prediction. *Appl. Soft Comput.* Elsevier. 49 1085–1093. DOI: 10.1016/j.asoc.2016.06.023

[113] B. Miranda and A. Bertolino. 2017. Scope-aided test prioritization, selection and minimization for software reuse. *J. Syst. Softw.* 131 (2017), 528–549. Elsevier. DOI: 10.1016/j.jss.2016.06.058

[114] O. Mizuno, E. Shigematsu, Y. Takagi, and T. Kikuno. 2002. On estimating testing effort needed to assure field quality in software development. In *Proceedings of the 13th International Symposium on Software Reliability Engineering*. IEEE. 139–146. DOI: 10.1109/ISSRE.2002.1173234

[115] N. A. Moketar, M. Kamalrudin, S. Sidek, M. Robinson, and J. Grundy. 2016. TestMEReq: Generating abstract tests for requirements validation. In *Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice*. ACM. 39–45. DOI: 10.1145/2897022.2897031

[116] R. Morales, A. Sabane, P. Musavi, F. Khomh, F. Chicano, and G. Antoniol. 2016. Finding the best compromise between design quality and testing effort during refactoring. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering*. IEEE. 24–35. DOI: 10.1109/SANER.2016.23

[117] S. Nageswaran. 2001. Test effort estimation using use case points. In *Proceedings of the Quality Week Conference*. Retrieved from http://www.bfpug.com.br/Artigos/UCP/Nageswaran-Test_Effort_Estimation_Using_UCP.pdf.

[118] M. Nasar and P. Johri. 2016. Testing resource allocation for fault detection process. In *Smart Trends in Information Technology and Computer Communications*. A. Unal et al. (Eds.). 683–690. DOI: 10.1007/978-981-10-3433-6_82

[119] G. C. Ndem, A. Tahir, A. Ulrich, and H. Goetz. 2011. Test data to reduce the complexity of unit test automation. In *Proceedings of the 6th International Workshop on Automation of Software Testing*. 105–106. DOI: 10.1145/1982595.1982618

[120] V. Nguyen, V. Pham, and V. Lam. 2013. qEstimation: A process for estimating size and effort of software testing. In *Proceedings of the International Conference on Software and System Processing*. ACM. 20–28. DOI: 10.1145/2486046.2486052

[121] H. Okamura, Y. Etani, and T. Dohi. 2011. Quantifying the effectiveness of testing efforts on software fault detection with a logit software reliability growth model. In *Proceedings of the Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*. 62–68. DOI: 10.1109/IWSM-MENSURA.2011.26

[122] F. Palomo-Lozano, A. Estero-Botaro, I. Medina-Bulo, and M. Núñez. 2018. Test suite minimization for mutation testing of WS-BPEL compositions. In *Proceedings of the Genetic and Evolutionary Computation Conference,* H. Aguirre (ed.). 1427–1434. DOI: 10.1145/3205455.3205533

[123] A. K. Pandey and N. K. Goyal. 2010. Test effort optimization by prediction and ranking of fault-prone software modules. In *Proceedings of the 2nd International Conference on Reliability, Safety and Hazard*, P. V. Varde et al. (Eds.). 136–142. DOI: 10.1109/ICRESH.2010.5779531

[124] R. M. Parizi, A. A. A. Ghani, S. P. Lee, and S. U. R. Khan. 2017. RAMBUTANS: Automatic AOP-specific test generation tool. *Int. J. Softw. Tools Technol. Transf.* 19, 6 (2017), 743–761. DOI: 10.1007/s10009-016-0432-3

[125] A. W. M. M. Parvez. 2013. Efficiency factor and risk factor based user case point test effort estimation model compatible with agile software development. In *Proceedings of the International Conference on Information Technology and Electrical Engineering (ICITEE'13)*. IEEE. 113–118. DOI: 10.1109/ICITEED.2013.6676222

[126] M. J. Pasha, S. Ranjitha, and H. N. Suresh. 2015. Testing-effort function for debugging in software systems and soft computing model. In *Proceedings of the International Conference on Green Computing and Internet of Things*. 913–919. DOI: 10.1109/ICGCIoT.2015.7380593

[127] N. Patel, M. Govindrajan, S. Maharana, and S. Ramdas. 2001. *Test Case Point Analysis: White Paper*. Cognizant Technology Solutions. Retrieved from https://www.cmcrossroads.com/sites/default/files/article/file/2013/XUS373692file1_0.pdf.

[128] R. Peng, Q. P. Hu, S. H. Ng, and M. Xie. 2010. Testing effort dependent software FDP and FCP models with consideration of imperfect debugging. In *Proceedings of the 4th International Conference on Secure Software Integration and Reliability Improvement* 141–146. DOI: 10.1109/SSIRI.2010.13

[129] R. Peng, Y. F. Li, W. J. Zhang, and Q. P. Hu. 2014. Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction. *Reliab. Eng. Syst. Safe.* 126 (2014), 37–43. DOI: 10.1016/j.ress.2014.01.004

[130] K. Periyasamy and X. Liu. 1999. A new metrics set for evaluating testing efforts for object-oriented programs. In *Proceedings of of the Conference on Technology of Object-Oriented Languages and Systems (TOOLS'99),* D. Firesmith (Eds.). 84–93. DOI: 10.1109/TOOLS.1999.787538

[131] I. Pinkster, B. van de Burgt, D. Janssen, and E. van Veenendaal. 2004. Successful test management: An integral approach. *Estimation* 85–112. Springer Science+Business Media. DOI: 10.1007/978-3-540-44735-1_6

[132] S. Ramacharan and K. V. G. Rao. 2016. Software effort estimation of GSD projects using calibrated parametric estimation models. In *Proceedings of the 2nd International Conference on Information and Communication Technology for Competitive Strategies*. DOI: 10.1145/2905055.2905177

[133] R. Ramler, C. Salomon, G. Buchgeher, and M. Lusser. 2017. Tool support for change-based regression testing: An industry experience report. In *Software Quality: Complexity and Challenges of Software Engineering in Emerging Technologies*, D. Winkler (Eds.). 133–152. DOI: 10.1007/978-3-319-49421-0_10

[134] S. S. Rathore and S. Kumar. 2017. Towards an ensemble based system for predicting the number of software faults. *Exp. Syst. Appl.* 82 (2017), 357–382. DOI: 10.1016/j.eswa.2017.04.014

[135] M. Reider, S. Magnus, and J. Krause. 2018. Feature-based testing by using model synthesis, test generation and parameterizable test prioritization. In *Proceedings of the IEEE 11th International Conference on Software Testing, Verification and Validation Workshops* 130–137. DOI: 10.1109/ICSTW.2018.00041

[136] A. Roman. 2015. Testowanie i jakość oprogramowania: Modele, techniki, narzędzia, Warszawa, Wydawnictwo Naukowe PWN, 2015, ISBN: 978-83-01-18160-4 (in Polish).

[137] D. S. Rosenblum and E. J. Weyuker. 1996. Predicting the cost-effectiveness of regression testing strategies. In *Proceedings of the 4th ACM SIGSOFT Symposiumon Foundations of Software Engineering*, D. Garlan (ed.). 118–126. DOI: 10.1145/239098.239118

[138] A. Rosenfeld, O. Kardashov, and O. Zang. 2018. Automation of Android applications functional testing using machine learning activities classification. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. 122–132. DOI: 10.1145/3197231.3197241

[139] A. Sadeghi, R. Jabbarvand, and S. Malek. 2017. PATDroid: Permission-aware GUI testing of Android. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*. 220–232. DOI: 10.1145/3106237.3106250

[140] A. Saeed, W. H. Butt, F. Kazmi, and M. Arif. 2018. Survey of software development effort estimation techniques. In *Proceedings of the 7th International Conference on Software and Computer Applications*. 82–86. DOI: 10.1145/3185089.3185140

[141] P. Sahoo, J. R. Mohanty, and D. Sahoo. 2018. Early system test effort estimation automation for object-oriented systems. In *Proceedings of the 6th International Conference on Frontiers of Intelligent Computing (FICTA'18)*. 325–333. DOI: 10.1007/978-981-10-7563-6_34

[142] P. Sahoo and J. R. Mohanty. 2017. Early test effort prediction using UML diagrams. *Indon. J. Electric. Eng. Comput. Sci.* 5, 1 (2017), 220–228. Institute of Advanced Engineering and Science. DOI: 10.11591/ijeecs.v5.i1.pp220-228

[143] M. Salmanoglu, T. Hacaloglu, and O. Demirors. 2017. Effort estimation for agile software development: Comparative case studies using COSMIC functional size measurement and story points. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*. 41–49. DOI: 10.1145/3143434.3143450

[144] S. Shamshiri, J. M. Rojas, J. P. Galeotti, N. Walkinshaw, and G. Fraser. 2018. How do automatically generated unit tests influence software maintenance? In *Proceedingss of the 11th International Conference on Software Testing, Verification and Validation*. 250–261. DOI: 10.1109/ICST.2018.00033

[145] A. Sharma and D. S. Kushwaha. 2011. A metric suite for early estimation of software testing effort using requirement engineering document and its validation. In *Proceedings of the 2nd International Conference on Computer and Communications Technology*. 373–378. DOI: 10.1109/ICCCT.2011.6075150

[146] A. Sharma and D. S. Kushwaha. 2013. An empirical approach for early estimation of software testing effort using SRS document. *CSI Trans. ICT* 1, 1 (2013), 51–66. DOI: 10.1007/s40012-012-0003-z

[147] A. Sharma and D. S. Kushwaha. 2012. Applying requirement based complexity for the estimation of software development and testing effort. *ACM SIGSOFT Softw. Eng. Notes* 37, 1 (2012), 1–11. DOI: 10.1145/2088883.2088898

[148] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan. 2013. Is lines of code a good measure of effort in effort-aware models? *Inf. Softw. Technol.* 55, 11 (2013), 1981–1993. DOI: 10.1016/j.infsof.2013.06.002

[149] T. Shippey, D. Bowes, and T. Hall. 2018. Automatically identifying code features for software defect prediction: Using AST N-grams. *Inf. Softw. Technol.* DOI: 10.1016/j.infsof.2018.10.001

[150] D. G. de Silva, B. T. de Abreu, and M. Jino. 2009. A simple approach for estimation of execution effort of functional test cases. In *Proceedings of the 2nd International Conference on Software Testing, Verification and Validation (ICST'09)*. 289–298. DOI: 10.1109/ICST.2009.47

[151] T. Silva-de-Souza and G. H. Travassos. 2017. Observing effort factors in the test design & implementation process of web services projects. In *Proceedings of the 2nd Brazilian Symposium on Systematic and Automated Software Testing*. DOI: 10.1145/3128473.3128480

[152] Y. Singh, A. Kaur, and B. Suri. 2008. An empirical study of product metrics in software testing. In *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*. M. Iskander (ed.). 64–72. DOI: 10.1007/978-1-4020-8739-4_12

[153] H. M. Sneed. 2018. Requirement-based testing—Extracting logical test cases from requirement documents. In *Software Quality: Methods and Tools for Better Software and Systems*, D. Winkler et al. (Eds.). 60–79. DOI: 10.1007/978-3-319-71440-0_4

[154] H. Srikanth, C. Hettiarachchi, and H. Do. 2016. Requirements based test prioritization using risk factors: An industrial study. *Inf. Softw. Technol.* 69 (2016), 71–83. DOI: 10.1016/j.infsof.2015.09.002

[155] P. R. Srivastava. 2009. Estimation of software testing effort: An intelligent approach. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE'09)*. https://www.researchgate.net/publication/235799428_Estimation_of_Software_Testing_Effort_An_intelligent_Approach.

[156] P. R. Srivastava. 2015. Estimation of software testing effort using fuzzy multiple linear regression. *Int. J. Softw. Eng. Technol. Appl.* 1, 2/3/4 (2015), 145–154. DOI: 10.1504/IJSETA.2015.075602

[157] P. R. Srivastava, A. Bidwai, A. Khan, K. Rathore, R. Sharma, and X. S. Yang. 2014. An empirical study of test effort estimation based on bat algorithm. *Int. J. Bio-Insp. Comput.* 6, 1 (2014), 57–70. DOI: 10.1504/IJBIC.2014.059966

[158] P. R. Srivastava, S. Kumar, A. P. Singh, and G. Raghurama. 2011. Software testing effort: An assessment through fuzzy criteria approach. *J. Uncert. Syst.* 5, 3 (2011), 183–201. ISSN: 1752-8909.

[159] P. R. Srivastava, A. Varshney, P. Nama, X. S. Yang. 2012. Software test effort estimation: A model based on cuckoo search. *Int. J. Bio-Insp. Comput.* 4, 5 (2012), 278–285. DOI: 10.1504/IJBIC.¸2012.049888

[160] Nanda S. Suharjito and B. Soewito. 2016. Modeling software effort estimation using hybrid PSO-ANFIS. In *Proceedings of the International Seminar on Intelligent Technology and Its Applications.* 219–224. DOI: 10.1109/ISITIA.2016.7828661

[161] R. T. Sundari. 2008. TCPA—Tool to test effort estimation. Retrieved from https://pdfs.semanticscholar.org/7ebf/62d5d9cc52f0acec5530212f557d50c1bca2.pdf.

[162] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, and S. H. Ameerjan. 2018. ESPRET: A tool for execution time estimation of manual test cases. *J. Syst. Softw.* 146 (2018), 26–41. DOI: 10.1016/j.jss.2018.09.003

[163] S. Tahvili, M. Saadatmand, M. Bohlin, W. Afzal, and S. H. Ameerjan. 2017. Towards execution time prediction for manual test cases from test specification. In *Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications.* 421–425. DOI: 10.1109/SEAA.2017.10

[164] S. Tahvili, M. Saadatmand, S. Larsson, W. Afzal, M. Bohlin, and D. Sundmark. 2016. Dynamic integration test selection based on test case dependencies. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops* 277–286. DOI: 10.1109/ICSTW.2016.14

[165] M. Thirasakthana and S. Kiattisin. 2018. Identifying standard testing time for estimation improvement in IT project management. In *Proceedings of the 3rd Technology Innovation Management and Engineering Science International Conference (TIMES-iCON'18).* 1–5. DOI: 10.1109/TIMES-iCON.2018.8621787

[166] F. Toure, M. Badri, and L. Lamontagne. 2018. Predicting different levels of the unit testing effort of classes using source code metrics: A multiple case study on open-source software. *Innov. Syst. Softw. Eng.* 14, 1 (2018), 15–46. DOI: 10.1007/s11334-017-0306-1

[167] S. N. Umar. 2013. Software testing effort estimation with Cobb-Douglas function: A practical application. *Int. J. Res. Eng. Technol.* 2, 5 (2013), 750–754. DOI: 10.15623/ijret.2013.0205003

[168] E. P. W. M. van Veenendaal and T. Dekkers. 1999. Testpointanalysis: A method for test estimation. In *Project Control for Software Quality.* R. Kusters et al. (Eds.). Retrieved from http://www.erikvanveenendaal.nl/NL/files/Testpointanalysis%20a%20method%20for%20test%20estimation.pdf.

[169] K. Wang, C. Zhu, A. Celik, J. Kim, D. Batory, and M. Gligoric. 2018. Towards refactoring-aware regression test selection. In *Proceedings of the 40th International Conference on Software Engineering.* 233–244. DOI: 10.1145/3180155.3180254

[170] Y. Wang, L. Wang, Y. Li, and X. Zhu. 2017. Time distribution of software stage effort. In *Proceedings of the 24th Asia-Pacific Software Engineering Conference Workshops.* 41–49. DOI: 10.1109/APSECW.2017.9

[171] Y. Wang, Z. Zhu, B. Yang, F. Guo, and H. Yu. 2018. Using reliability risk analysis to prioritize test cases. *J. Syst. Softw.* 139 (2018), 14–31. DOI: 10.1016/j.jss.2018.01.033

[172] A. C. Y. Wong, S. T. Chanson, S. C. Cheung, and H. Fuchs. 1997. A framework for distributed object-oriented testing. In *Formal Description Techniques and Protocol Specification, Testing and Verification*, T. Mizuno et al. (Eds.). 39–56. DOI: 10.1007/978-0-387-35271-8_3

[173] H. Wu, C. Nie, and F. C. Kuo. 2016. The optimal testing order in the presence of switching cost. *Inf. Softw. Technol.* 80 (2016), 57–72. DOI: 10.1016/j.infsof.2016.08.006

[174] D. K. Yadav and S. Dutta. 2016. Test case prioritization technique based on early fault detection using fuzzy logic. In *Proceedings of the 10th 3rd International Conference on Computing for Sustainable Global Development (INDIACom'16).* 1033–1036.

[175] S. Yamada, J. Hishitani, and S. Osaki. 1993. Software-reliability growth with a Weibull test-effort: A model & application. *IEEE Trans. Reliab.* 42, 1 (1993), 100–106. DOI: 10.1109/24.210278

[176] S. Yamada, H. Ohtera, and H. Narihisa. 1986. Software reliability growth models with testing-effort. *IEEE Trans. Reliab.* 35, 1 (1986), 19–23. DOI: 10.1109/TR.1986.4335332

[177] M. Yan, Y. Fang, D. Lo, X. Xia, and X. Zhang. 2017. File-level defect prediction: Unsupervised vs. supervised models. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* 344–353. DOI: 10.1109/ESEM.2017.48

[178] J. Yang, J. Chen, W. Hu, and Z. Deng. 2017. Web-based software reliability growth modelling for mobile applications. In *Proceedings of the 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP'17).* 342–346. DOI: 10.1109/ICCWAMTIP.2017.8301510

[179] J. Yang. 2017. Estimating effort of test automation projects. In *uTest, Applause App Quality.* Retrieved from https://www.utest.com/articles/estimating-effort-of-test-automation-projects.

[180] J. Yang, R. Wang, Z. Deng, and W. Hu. 2011. Web software reliability analysis with yamada exponential testing-effort. In *Proceedings of the 9th International Conference on Reliability, Maintainability and Safety*. 760–765. DOI: 10.1109/ICRMS.2011.5979367

[181] H. Younessi, P. Zeephongsekul, and W. Bodhisuwan. 2002. A general model of unit testing efficacy. *Softw. Qual. J.* 10, 1 (2002), 69–92. DOI: 10.1023/A:1015724900702

[182] C. M. Zapata-Jaramillo and D. M. Torres-Ricaurte. 2014. Test effort: A pre-conceptual-schema-based representation. *DYNA* 81, 186 (2014), 132–137. National University of Colombia, Medellín, Mines Faculty. DOI: 10.15446/dyna.v81n186.39753

[183] X. Zhu, B. Zhou, L. Hou, J. Chen, and L. Chen. 2008. An experience-based approach for test execution effort estimation. In *Proceedings of the 9th International Conference for Young Computer Scientists*. 1193–1198. DOI: 10.1109/ICYCS.2008.53

[184] X. Zhu, B. Zhou, F. Wang, Y. Qu, and L. Chen. 2008. Estimate test execution effort at an early stage: An empirical study. In *Proceedings of the International Conference on Cyberworlds*. 195–200. DOI: 10.1109/CW.2008.34

[185] F. Zou and R. Xu. 2004. Empirical measurement of the software testing and reliability. *Wuhan Univ. J. Nat. Sci.* 9, 1 (2004), 23–26. DOI: 10.1007/BF02912711

Received December 2019; revised September 2020; accepted December 2020