Software Change Size Measurement: An Exploratory Systematic Mapping Study

Tuna Hacaloglu[1,2], Neslihan Küçükateş Ömüral[3], Görkem Kılınç Soylu[4,5], Onur Demirörs[4]

[1] *École de Technologie Supérieure, 1100, rue Notre-Dame Ouest, Montréal, Québec, H3C 1K3, Canada*
[2] *Atilim University, İncek, Gölbaşı, Ankara, 06830, Türkiye*
[3] *Middle East Technical University, Üniversiteler, Çankaya, Ankara, 06800, Türkiye*
[4] *Izmir Institute of Technology, Gülbahçe, Urla, İzmir, 35430, Türkiye*
[5] *Izmir University of Economics, Sakarya Caddesi No:156, Balçova, İzmir, 35330, Türkiye*

Abstract

Change in software projects can occur through various channels. Customers may request modifications or new features; appraisal activities such as reviews or testing may uncover issues that necessitate adjustments, or products may need to adapt to changes in their operating environment. Therefore, it is essential to assess these changes explicitly and objectively within the scope of software engineering activities. Specifically, quantifying change by measuring its size is crucial for successful management, as without a meaningful metric, it is impossible to accurately assess its impact on the project's effort, schedule, and cost. This study aims to explore the concept of change in software engineering literature, with a particular emphasis on the methods used to measure its size. The study reveals that the current literature on this topic is still in its early stages and the measurement and estimation of changes remain challenging throughout both development and maintenance phases. According to the reviewed articles, size is primarily used for effort estimation. Various software artifacts from different stages of the Software Development Life Cycle (SDLC) serve as input for change measurement, highlighting the need for a versatile size measurement applicable across all SDLC phases. Most of the reviewed articles interpret change in the context of maintenance activities. This research sets a benchmark for the status of software size measures for software change and highlights related problems to suggest further research topics.

Keywords

Size measurement, software change, software maintenance

1. Introduction

The common goal of all software engineering activities is to produce high-quality software. Broadly defined, software quality means delivering a product that meets requirements while adhering to the negotiated schedule and budget. Although software requirements are ideally specified early in the development process, the dynamic nature of software development makes change inevitable. Welcoming change became a key principle in the Agile Manifesto in 2001[1]. Since then, change management has gained significant attention and is now a well-recognized practice in the software development community. “Change” in the software engineering domain can be interpreted differently according to the point at which it occurs during two major phases of software life cycle: development and maintenance.

Change can happen during development as customer "change requests" or from quality assurance activities like reviews or testing. If change occurs after deployment, it falls under maintenance activities. All types of change are important and should be considered in project

IWSM-Mensura, September 30–04, 2024, Montréal, Canada
tuna.hacaloglu@etsmtl.ca (T. Hacaloglu); neslihan.omural@metu.edu.tr (N. Küçükateş Ömüral); gorkem.soylu@iue.edu.tr (G. Kılınç Soylu); onurdemirors@iyte.edu.tr (O. Demirörs)

management. Measuring the magnitude of change is crucial. Software size, an important input for software estimation, is among the most critical measurable attributes. Without an appropriate size measure, planning, estimating, and controlling large-scale projects objectively become highly challenging [2].

Furthermore, size-based assessments provide managers with opportunities to manage, maintain, and improve projects, as well as to compare them to each other [3]. Therefore, measuring the size of software change can assist in making accurate estimates to control the impact of change, thereby preventing project schedule and budget overruns. The measurement of software size and its benefits in estimating software development is a topic that has been extensively researched. However, measuring the size of change remains debatable due to different levels of abstraction of change and when it happens in the software development life cycle. This study aims to contribute to this field by focusing on measuring "software change." We conducted an exploratory systematic mapping study to delve into this topic. We analyzed 25 studies based on objective size measure, measurement input, aim of measurement, type of assessment for estimations, estimation accuracy metrics, domain and dataset types, type of SDLC, challenges mentioned, and the year the study was conducted. Key findings reveal that size is commonly used to measure software change, primarily for effort estimation. Change is mostly related to maintenance activities in which the authors highlight the lack of a well-established estimation method for maintenance tasks. Various software artifacts (requirements, design documents, source code) from different stages of the software development life cycle (SDLC) are used for measuring change. Regression analysis is frequently employed to create prediction models. These studies frequently neglect crucial statistical tests, like assessing normal distribution, necessary for dataset suitability in regression analysis. This violation of traditional statistical assumptions results in inaccurate regression analysis in software estimation, highlighting the need for further research to find solutions. Exciting research opportunities involve investigating size measurement techniques for changes in object-oriented software, web applications, and projects facing technological shifts, especially in Agile environments and during refactoring. Additionally, exploring the adaptability of size measurement methods in different types of software maintenance shows promise for further study.

The rest of this article is organized as follows: section 2 presents the background, section 3 describes the research methodology, section 4 presents the research results, section 5 presents a discussion on the findings, and section 6 presents the conclusion and recommendations for future work.

2. Background

Software measurement standards offer recommendations for measuring these changes. According to the COSMIC Functional Size Measurement (FSM) [11], a 'functional change' in existing software means adding new data movements, modifying existing ones, or deleting them, including associated data manipulation. The size of these changes is calculated by adding the size of added, changed, and deleted data movements. This size is measured in COSMIC Function Points (CFP). In [6], the authors assessed the usage of the COSMIC FSM method for sizing software changes. They achieved an effort estimation accuracy of 10 to 20 percent for software changes of a client's project by measuring size as COSMIC Function Points (CFP). IFPUG 4.1 [12] calculates the enhancement project function point count similarly by the sum of function points of functions to be added, deleted, and changed multiplied by a value adjustment factor. NESMA [13] defines an enhancement project as a project in which enhancements are carried out on an existing application. The functional size of an enhancement project is determined by the sum of function points of the transactions and/or logical files to be added, deleted, and changed. NESMA

has published a guide "FPA for software enhancement" [14] that describes a method for sizing enhancement projects which is different from the method in the International Standard. This guideline considers the impact factor of the changes. During the calculation of size of an enhancement project, it proposes to multiply the function points of the data functions and transactional functions to be changed by their determined impact factors. In [7] the authors suggested measuring “changes” where pre-built functionality is not sufficient for the customer requirements.

3. Research Methodology

The review process was performed according to the guidelines defined by Kitchenham [4]. Our review protocol includes the following steps: identification of research questions, selection of primary studies, the definition of quality assessment criteria, performing a quality analysis, data extraction, and monitoring and data synthesis. To minimize the risk of misunderstandings or bias, we performed peer review in both quality assessment and data extraction steps. In the following, we present the description of the steps involved in more details. In this study, our goal is to investigate literature on measuring software change size. To achieve this, we've formulated following research questions.

- RQ1. How are publications distributed over the years?
- RQ2. Which size measurement/estimation methods were developed or suggested to “measure the size of change in a software”?
    - RQ2.1. Which software artifacts are used as inputs for size measurement?
    - RQ2.2. What is the aim of using these size measurement methods?
- RQ3. How are these size measurement methods validated?
- RQ4. What are the challenges reported for measuring the size of change?

For our data sources, we have chosen Web of Science and Scopus for conducting our searches. We limited our search to articles published after 2000. The search string used in this study is as follows:

software & (size OR sizing OR effort OR measur* OR estimat* OR predict*) & (change OR reuse OR maint*)

Our search resulted in 505 articles after eliminating the duplicates. The initial set of 505 articles were examined with respect to the inclusion and exclusion criteria. The main inclusion criterion of our review is that the paper describes “research on measuring/estimating the size of change in a software”. Studies related to effort, and size/cost factors are only included if they are related to size measurement. The studies must be in the software engineering domain to be included. Books, theses, and gray literature are excluded in our review process. Studies that are not in English and studies for which the full text is not available are also excluded. According to our research purpose, papers not directly related to measuring the size of change and overlapping papers describing the same study are excluded.

Furthermore, papers having missing information in the definition of the methods or validation process are also excluded. The examination was done mainly by reading the title and abstract and by fast reading the article when title and abstract did not provide enough information for deciding. At the end of this step 83 articles were selected. We have also done forward and backward snowballing to extend the main database search and capture any relevant ones not returned by our search, we have performed snowballing [5] for 10 randomly selected papers and 1 hand-picked paper, which we identified as the most relevant article to our scope with a high number of citations. In the forward snowballing step, we included Google Scholar for finding the citing articles of a specific article. Thus, at the end of snowballing, we had 94 articles in our list. The following quality assessment criteria are designed to refine our search results by eliminating out-of-scope, scientifically immature, incomplete, and unsatisfactory studies.

QA1. Is there a clear statement that the study proposes or uses the size measurement for the project change management?

QA2. Can it be inferred from the study that the size measurement method is newly proposed, or is it only applied as already available?

QA3. Are the limitations and assumptions of the study stated explicitly?

94 articles were examined with respect to the above listed quality assessment criteria and the assessment resulted in a final set of 25 articles.

4. Research Results

4.1. The distribution of publication across years

Answers for the RQ1 are presented in this section. The distribution of the primary papers by years is given in Figure 2. As can be seen in the Figure 1, there is no clear increase or decrease pattern in the distribution of studies over the years. According to this distribution, the most publications were made between 2011 and 2013.

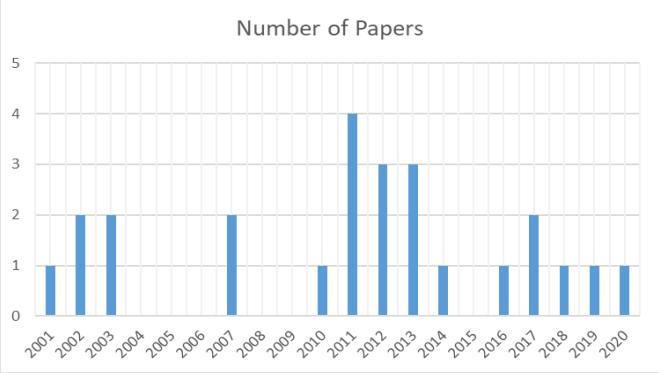

Figure 1: The distribution of the papers over years

4.2. Observed size measurement/ estimation practices

Regarding the RQ2 and RQ3, all primary papers were deeply analyzed to evaluate the size measurement/estimation methods developed or suggested to "measure the size of change in a software".

4.2.1. Methods employed for software change size measurement

Regarding the RQ2, all primary papers were deeply analyzed to evaluate the size measurement/estimation methods developed or suggested to "measure the size of change in a software". Objective size measures in these papers are listed in Table 1. According to this list, size measures "Function Points (FP)" and "Source Line of Code (SLOC)" seem to dominate this field; 9 of the papers suggest using SLOC, and 11 of the papers suggest sizing as FP. In function point size measurement papers, COSMIC functional size measurement is suggested more than IFPUG functional size measurement. All the papers except [S19] suggest objective size measures, in that paper "story points" were suggested for measuring change.

Table 1 Objective Size Measures

Objective Size Measure	Article
# Functions (edited, forward, backward)	S6
CC (Class & Complexity)	S11, S25
CISE (Change Impact Size Estimate)	S18
COSMIC Function Point	S1, S3, S4, S5, S17, S21, S23
IFPUG Function Point	S14, S15, S20, S22
MS-MC (# Methods per Class)	S8
SLOC (Source Lines of Code)	S2, S7, S8, S10, S12, S13, S16, S24, S25
SNAP (Software Non-functional Assessment Process)	S20
SSM (Structural Size Measure)	S1

4.2.2. Software artifact types used in software change size measurement

Considering the RQ2.1, different types of software artifacts were used as input for sizing in these papers, most used ones were requirement documents, design documents and source code of the software. To answer the RQ2.2, the aim to use size measurement methods in each paper was analyzed. The results are presented in Table 2. The results show that "effort estimation" is the most common aim for measuring size.

Table 2 Size Measurement Aims

Aim of Measuring Size	Article
assuring adaptive maintenance	S25
assuring business relationship with suppliers	S14
change impact size estimation	S18
cost estimation	S16
effort estimation	S2, S3, S5, S6, S8, S9, S10, S13, S15, S17, S19, S20, S22, S23, S24, S25
maintainability measurement	S3
maintenance type prediction	S12
prioritizing and evaluating CRs	S1
securing software quality	S11
size measurement	S7, S21

4.2.3. Methods Employed in Validating Measurements

These size measurement methods were validated by using different types of assessments. Regarding the RQ3, the type of assessments in these papers are listed in Table 3. Considering the results, "regression analysis" is by far the most frequently used method.

Table 3 Assessment Types

Type of the assessment	Article
Bayesian analysis	S24
Linear discriminant analysis	S2
Multivariate linear regression analysis	S16
One-sided Mann–Whitney U test	S12
Regression analysis	S4, S6, S7, S8, S9, S10, S13, S15, S17, S20, S23, S25

As presented in Table 4, MMRE and PRED(N) are the most frequently used accuracy assessment metrics in these papers. MMRE is "Mean Magnitude of Relative Error" and PRED (N) represents “the percentage of observations with an MRE less than or equal to N”. In the paper [S16], three different metrics PRESS, SPR and MdMRE are used. PRESS is defined as the sum of the squared prediction errors, SPR is defined as the sum of the absolute value of the PRESS residuals and MdMRE represents “Median Magnitude Relative Error”.

Table 4 Assessment Accuracies

Assessment Accuracy	Article
Correlation Coefficients	S4, S9, S10, S20, S23
MdMRE	S16
MMRE	S2, S5, S7, S8, S12, S13, S16, S18, S23, S25
PRED(N)	S7, S12, S13, S16, S20, S23
PRESS	S16
p-value	S6, S20
SPR	S16

The results showed a wide variation in accuracy in these evaluations. MMRE values showed significant fluctuations, ranging from a low 1.12% to a high 115.42%. This highlights the differences in how well the predictions aligned with the actual outcomes across various studies. Both low and high ends of the interval were observed in paper [S7], in which two approaches are used to estimate change size in packages throughout the releases. One is based on structural properties (SP) whereas the other one is based on historical changes (HC). MMRE values for SP models exceeded 60%, even reaching 115%, whereas the highest MMRE value observed with HC models is 49.92%, and the smallest MMRE value is 1.12%. Based on the observed values, the authors argue that HC models show superior performance compared to SP models.

As seen in Table 4, PRED(N) is the second most used assessment type. Similarly to MMRE, PRED(N) values vary over a wide range. The lowest calculated value is 3.85% for PRED(25) whereas the highest value 100% was observed for PRED(20), for PRED(25) and for PRED(30) in different papers. In [S7], very low PRED(25) values were observed (4.35%, 12.50%, 3.85% and 9.52%). As discussed above, the paper examines two approaches to estimate change size in packages throughout the releases. The method based on structural properties (SP) showed low accuracy and this was reflected by the low PRED and high MMRE values. The highest values of PRED(N) were observed in paper [S20] which explores using Software Non-Functional Assessment Process (SNAP) together with FP for effort estimation. The authors distinguished adding new functions and modifying existing functions in the analysis. PRED(25) and PRED(30) values were 100% when using the FP based model for effort estimation for adding new functions

whereas the value of PRED(20) was 81.818%. PRED(20) increased to 100% when SNAP analysis was used together with the FP based model.
The PRED values of the FP based model were lower for modifying existing functions, namely 94.737% for PRED(20), PRED(25) and PRED(30). PRED(30) value for modifying existing functions increased to 100% from 94.737% when the authors used SNAP analysis together with FP, however PRED(20) decreased to 84.211% whereas PRED(25) remained the same. The authors deduced that SNAP is not an effective size measure for adding new functions. Using both FPs and SNAP improved the effort estimates for adding new functions, but neither improved nor worsened the estimates for projects modifying existing functions. In these validations, different types of domains were used; all the used domains are listed in Table 5. The results show that software development & technology, banking & commerce and business & management are the most used domains for validating the methods.

Table 5 Domains of the articles

Domain	Article
Software Development and Technology	S2, S6, S7, S8
Online Services	S3, S4
Automotive Software	S11, S18
Banking and Commerce	S1, S14, S22
Defense	S17
Business and Management	S15, S16, S19
Telecommunication Applications	S23
Language and Linguistics	S17
Not specified	S5, S9, S10, S12, S13, S20, S21, S24

The dataset for validation was also analyzed in these papers. Most of the papers used industrial datasets, as seen in Table 6.

Table 6 Datasets of the articles

Dataset	Article
Academia	S9, S12, S13, S18, S21, S25, S20
Industrial	S1, S2, S3, S4, S5, S6, S7, S8, S10, S11, S13, S15, S16, S17, S19, S21, S22, S23, S24, S25

4.2.4. Challenges mentioned in the articles

The articles were examined considering researchers' identified challenges to pinpoint areas requiring further investigation in software change measurement studies. As a result, the difficulties listed in Table 7 were identified, and a detailed discussion regarding them was presented in the "Discussion" section.

Table 7 Challenges mentioned in the articles

Challenge	Article
Difficulty and lack of metrics and methods for estimating effort in maintenance tasks	S2, S5, S19, S24
Lack of historical data to apply analogy/expert-based approaches for effort estimation	S19
Problems with functional size measurement	S22, S23, S25
Problems with LOC for maintenance	S8
Incomplete artifacts	S18

5. Discussion

5.1. Size measurement of Software Change

It is noticeable that software size measurement has established its relevance in quantifying the magnitude of the change. When the articles are further analyzed, it is observed that pure size measurement of the change is studied in only three articles [S4], [S7], [S21]. On the other hand, in 22 articles change size measure is used as input for various estimations, including effort [S2], [S5], [S6], [S8], [S9], [S14], [S15], [S16], [S17], [S19], [S20], [S22], [S23], [S24], [S25] and cost [S16] and for the aim of assuring the business relationship with suppliers [S14].
Regarding size measure, research studies can be broadly divided into three major categories: measuring the requirements using functional size measurement (IFPUG and COSMIC), measuring the code length using line of code (LOC), and measuring the object-oriented code size using the number of methods per class or class complexity.
Accordingly, the input software artifacts that are used in size measurement belong to different phases of SDLC: user stories, use cases, requirements in the requirement analysis phase; program code during the implementation phase; change specifications, and maintainability requirements in the maintenance phase. These artifacts serve as key indicators of the occurrence of software changes across different SDLC phases and emphasize the significance of a tailored method for size measurement, particularly in the context of software changes.

5.2. Estimations with change size

Out of the 25 articles reviewed, 13 of them explicitly mention the use of regression analysis as a method for developing effort estimation models with software change size as an independent variable. However, these studies appear to neglect essential prerequisite statistical tests, such as those assessing normal distribution, to ascertain the suitability of their datasets for regression analysis. This finding can indicate a misalignment among the software engineering dataset characteristics and conventional statistical data analysis practices. Hence, one could argue that traditional statistical analysis assumptions are potentially being violated, and regrettably, there appears to be an incorrect application of regression analysis in the context of software estimation. This issue poses a noteworthy challenge, calling for further research to uncover the underlying causes and to find out potential remedies.

Another noteworthy observation is the omission of any mention in the articles within the pool regarding the chosen software development life cycle model employed in the projects. Although this is a factor which seems to vary considerably for managing software changes, this information is absent in the discussions. Only Scrum [S1], XP [S19], and Rational Unified Process [S19] were the stated SDLC models. The importance of this situation becomes clear when comparing the traditional Waterfall model with the Agile approach. Within Agile software development, a fundamental principle is the execution of iterative and incremental development,

characterized by an open embrace of change. Conversely, in a traditional model such as the Waterfall Model, the commencement of a particular phase is contingent upon the full completion of its predecessor. As the phases are frozen upon their completion, the Waterfall Model exhibits a limited enthusiasm for accommodating change. Nevertheless, as it is practically impossible to avoid change in the landscape of project management, it would be valuable to mention the SDLC models employed in these studies to appraise the efficacy of software size measurement within various SDLC models and to explore how the chosen SDLC model handles software changes. This finding serves as an indication that further research is needed regarding software size measurement, specifically in the context of change management, across various software development life cycle models.

5.3. The Significance of Maintenance Phase in Change Size Measurement

Upon the analysis of all 25 articles, it is observed that the concept of software change is predominantly addressed within the framework of maintenance activities in 20 of these articles and a significant portion of the articles used size measurement for estimation purposes.

Authors frequently emphasize that the maintenance phase is widely recognized as the most effort demanding and costly stage within the software development life cycle [S4], [S6], [S12], [S19], [S24]. In [S6] based on previous research results by [8]. The authors point out that this is a consequence of the substantial commitment of maintenance efforts, time, and resources essential for effectively resolving issues during software maintenance activities and they put forward that inadequate handling of these endeavors can result in a decline in the software's maintainability. Regarding this point of view, the references given in [S6] are quite old and new research is required to draw more sound conclusions. The authors in [S6] identified high maintenance efforts by examining the distribution of all maintenance activities within the software project and concluded that these high-maintenance areas typically resulted in significant changes to the source code due to the correlation result obtained between high maintenance effort and code changes with high complexity metrics.

In [S12] the authors investigated the impact of different Lines of Code (LOC) metrics on maintenance effort. The study's findings reveal that the program consumes a significant portion, up to 50%, of the effort during corrective maintenance. The study highlights the practicality of using metrics related to LOC that are added, modified, and deleted as reliable indicators for software maintenance cost estimation. Effort estimation models for maintenance tasks could be improved by treating these LOC metrics as separate parameters rather than merely adding them together. However, we must emphasize that since LOC is not an upfront size metric to be utilized for estimation related purposes, there should be further research to make estimations when a change request arrives.

When examining articles that assert a higher unit cost for maintenance effort compared to development effort, it becomes evident that the claim of maintenance being more costly is rooted in prior research findings in the literature. Most of the articles in the dataset, however, do not directly compare the unit cost of maintenance with development to substantiate this conclusion. The primary reason for this gap is the absence of a standardized unit of measurement for determining maintenance unit costs.

Another challenging category synthesized from the articles in the pool is the view of a lack of a well- established estimation method for maintenance tasks [S5], [S19], [S24]. In addition, in [S2] the authors draw attention to the extreme difficulty of estimating the software maintenance accurately and reliably both in academia and industry. This point of view is reinforced in [S24], where the authors state that this challenge exists due to the lack of metrics and suitable methods in software maintenance effort estimation. On the other hand, authors in [S4] state that cost of maintenance is more than development cost and point out that the computation of function points in a maintenance project differs from that in a development project. According to [S4] in

COSMIC-FFP, new developments count all messages in a sequence diagram as function points, while maintenance projects exclude messages to unchanged objects from the count. This distinction is essential to differentiate between future maintenance costs and past development costs. The authors in [S4] created a tool to calculate COSMIC-FFP function points from source code undergoing maintenance. They conducted regression analysis to assess the relationship between COSMIC-FFP function points and the LOC in a real maintenance project. The findings indicate a strong correlation between COSMIC-FFP function point values and LOC, affirming the effectiveness of COSMIC-FFP as a reliable measure of software size for this specific maintenance project. However, as in [S12], utilization of COSMIC by taking the source code as an input for measurement is not an upfront size metric to be able to do effort estimation and more studies are required to perform estimations when a change request arrives.

In [S19], while supporting the view of high effort and resource demanding characteristics of software maintenance, the authors draw attention to the existence of fewer methods for maintenance effort estimation. More specifically, the lack of a standardized size measurement method in software maintenance projects appears as a challenge within the context of agile software development with the inapplicability of frequent effort estimation approaches such as poker planning, due to the unavailability of historical data to apply such analogy/ expert opinion-based methods and yields unrealistic results [S19]. The authors in [S19] proposed an effort estimation model where they define the maintenance size as Adjusted Story Points (ASP) which is a combination of Value adjustment factors composed of documentation quality, structuredness, modularity, and reusability and story points. Nevertheless, the article lacks evidence demonstrating the effectiveness of this size metric in estimating effort. It criticizes the use of analogy-based estimation in agile maintenance projects but employs another subjective estimate, namely "story points," to define the size of maintenance software.

5.4. Criticism of Literature Regarding Change Size Measurement Methods

Available traditional size measurement methods are criticized in the articles of the pool in different aspects which are presented in the details in the following paragraphs. For instance, in [S23] the authors criticize that available size measurement methods such as COSMIC FSM don't consider software complexity and emphasize the need for an effort and cost estimation model containing not only the size but also the complexity. From their paper, we understand that the complexity dimension of the size perhaps be considered to generate sound effort estimations for software change. In [S22], the authors oppose the utilization of function points for estimating maintenance tasks of Web applications. The authors in [S22] argue that even though functional size measurement is said to be technology independent, the count of function points varies based on the database and data communication systems in use. Web-based systems yield different function-point counts compared to client/server or monolithic mainframe systems. Relational databases with normalization result in more function points than hierarchical file systems. The choice of programming languages and technologies significantly influences the definition of inputs, outputs, interfaces, and databases. For instance, inputs and outputs in a web-based system exhibit notable distinctions when compared to those in a mainframe online system or a client/server system. Therefore, the idea of requiring size measurement for innovative architectures also applies to maintaining new generation projects [9].

The authors in [S8] criticize the utilization of LOC during the maintenance process due to difficulty of upfront estimation with LOC. The authors in [S8] point to the fact that at present, the size of software maintenance is determined solely by the LOC that undergoes changes during the maintenance process. However, since the software is developed using an object-oriented language, it is essential to consider classes and methods as well and criticizes utilizing LOC for predicting maintenance time in advance and proposed to utilize design class diagrams for new software maintenance size metrics based on number of classes and methods changed. Another

study related to object-oriented software maintenance conducted by the authors in [S25] who pointed out that object-oriented software projects need adaptive maintenance and pointed out that the importance of considering "is-part-of, is-referred-by, and is-a" aspects of object-orientation. Specific metrics are needed to quantify these factors; otherwise, the associated expenses, such as specialization costs, object reuse costs, development costs, reengineering costs, maintenance costs, and so on, cannot be easily measured. They reported based on previous studies that traditional metrics such as Halstead measure, LOC, McCabe's Cyclomatic Complexity are not considering the object-oriented aspects. In their study they found that during adaptive maintenance, local attributes tend to have less significance as they usually involve the addition of only a few attributes. In contrast, inherited attributes are crucial, as they serve to conceal internal states for instances of subclasses.

Considering [S8] and [S25], two potential avenues for future research emerge. One such direction involves conducting further studies to better understand the measurement of change size in object-oriented software. The other direction can be related to focusing more on the different types of maintenance related to sizing software change. In the literature maintenance can be classified into three major categories such as corrective, adaptive, and perfective maintenance [10]. However, in our pool we saw that only a few articles handled software size measurement for different types of maintenance. This can be a future research direction to analyze how size measurement on the changes appearing different types of maintenance and to identify potential improvement points. All these challenges may be due to the problems that arise while dealing with "change" related issues during the development or maintenance stage. In [S1] the authors draw attention to the lack of change evaluation process and show this fact as the reason for unsatisfactory results of several agile projects. This is interesting since, "welcoming change" is one of the four principles of Agile Manifesto [1] and constitutes the core of agile software development since 2001. Therefore, the reasons behind the deficiency in change evaluation should be further researched.

In [S5] the authors pointed out that even though software projects are largely dominated by maintenance; there has been insufficient explanation of the fundamental size measurement technologies used for estimation thus far. The authors in [S18] highlighted the challenge of the change acceptance decision for requested changes in the software development phase and the fact that change request impact assessment is based on incomplete software artifacts. Authors in [S24] support this view by emphasizing that it is significantly more difficult to gather maintenance data because it needs more specific sizing criteria and because enterprises sometimes do not have a software maintenance process that is as strict as its development equivalent to enforce the practice of data gathering and analysis.

The lack of historical data is also shown as a barrier to apply analogy-based, expert-opinion based estimation for maintenance projects in [S19]. Based on the claim of the authors in [S18], we can show the incomplete artifacts among the obstacles in measuring and/ or estimating the change because we anticipate that incomplete artifacts may not contain enough details to apply a size measurement method such as COSMIC FSM. Therefore, there should be a mature description of artifacts to be an input for size measurement and historical database to be able to make comparisons. This situation requires further investigation with case studies.

To sum up, sizing and/or estimation remain a challenge with the scope of dealing with change, independent of development or maintenance phase. There are several claims in the articles, but these claims are not much evaluated with sound validations. There is a need for a more in-depth exploration of the distinctive aspects of software change in various project types, such as complex web applications composed of numerous services and object-oriented software. Furthermore, as discerned from the articles, exploring different types of maintenance could be an interesting research domain for advancing our understanding of software sizing. Finally, the software change size related studies comprise 20 years of continuous work. Even though there

are few researchers focusing on the domain within this year range, the related literature is not mature yet.

6. Conclusion and Future Work

In this study, we aimed to take a closer look at the change concept in software engineering from the size measurement perspective. After observing 25 articles, main findings of the study are as follows: in majority of the articles change is interpreted related to maintenance activity. In most of the articles in the pool, size is used for the purpose of effort estimation. Most used size attributes in these articles are functionality, length and number of methods and class complexity for object-oriented code. Different software artifacts such as requirements, design documents, source code from different stages of SDLC are used as input for measurement of change, which shows the importance and need of a size measurement of change that would be applicable in these different SDLC phases. Like typical software estimation studies, regression analysis is used to create prediction models in this domain the most. This study has identified promising directions for future research. These include investigating size measurement techniques for assessing changes in object-oriented software, web applications, and emerging software projects, particularly when technological shifts occur in Agile projects and during the process of refactoring. Furthermore, there is potential to explore the adaptability of size measurement methodologies in various software maintenance types.

In conclusion, the measurement and estimation of software size changes during both development and maintenance phases continue to present significant challenges. The existing body of related literature is still in its nascent stages, demanding further research. We anticipate that this study will offer valuable insights and pave the way for future investigations. As a prospective endeavor, expanding the review with more databases and conducting interviews with industry practitioners to gather their feedback on this matter could provide valuable perspectives and contribute to the ongoing discourse.

Acknowledgements

We would like to thank the support of the Scientific and Technological Research Council of Turkey (TUBITAK) ARDEB 1001 [Project number: 121E389] program.

References

[1] https://agilemanifesto.org/iso/en/manifesto.html

[2] C. Gencel and O. Demirors, “Functional size measurement revisited,” ACM Trans. Softw. Eng. Methodol., vol. 17, no. 3, pp. 1–36, 2008.

[3] A. Abran, C. Symons, C. Ebert, F. Vogelezang, and H. Soubra, “Measurement of software size: Contributions of cosmic to estimation improvements,” in The International Training Symposium, At Marriott Bristol, United Kingdom, 2016, pp. 259–267.

[4] B. Kitchenham and S. Charters. 2007. Guidelines for performing systematic literature reviews in software engineering.

[5] C. Wohlin. 2014. “Guidelines for snowballing in systematic literature studies and a replication in software engineering.”

[6] C. Ebert and H. Soubra, “Functional size estimation technologies for software maintenance,” IEEE Softw., vol. 31, no. 6, pp. 24–29, 2014.

[7] N. Küçükateş Ömüral, O. Demirörs, “A Size Measurement Method for Enterprise Applications,” in Proceedings of the Joint Conference of the 31st International Workshop

on Software Measurement (IWSM) and the 16th International Conference on Software Process and Product Measurement (MENSURA), 2022.
[8] LJ Arthur, "Software evolution: The Software Maintenance Challenge," Wiley-Interscience: New York, NY, 1988. Yip S, Lam T. A software maintenance survey. Proc APEC'94, Dec 1994; 70–79.
[9] T. Hacaloğlu, and O. Demirörs, "An exploratory case study using events as a software size measure," Information Technology and Management, 2023, 1-20.
[10] D. Galin, "Software quality assurance: from theory to implementation," Pearson education, 2004.
[11] ISO/IEC 19761: 2011, Software Engineering – COSMIC: A functional size measurement method, International Organization for Standardization, Geneva, 2011
[12] ISO, I. (2003). IEC 20926: Software Engineering-IFPUG 4.1 Unadjusted Functional Size Measurement Method-Counting Practices Manual. International Organization for Standardization, Geneva, Switzerland.
[13] NESMA, D. (1997). Counting Guidelines for the Application of Function Point Analysis.
[14] Engelhart, J., & Langbroek, P. (2009). Function point analysis (FPA) for software enhancement. Nesma.

[S1] Hakim, H., Sellami, A., & Ben-Abdallah, H. (2020). An in-Depth Requirements Change Evaluation Process using Functional and Structural Size Measures in the Context of Agile Software Development. In ICSOFT (pp. 361-375).
[S2] Singh, C., Sharma, N., & Kumar, N. (2019). An Efficient Approach for Software Maintenance Effort Estimation Using Particle Swarm Optimization Technique. International Journal of Recent Technology and Engineering (IJRTE), 7(6C), 1-6.
[S3] Almakadmeh, K., Al-Sarayreh, K. T., & Meridji, K. (2018). A Measurement Model of The Functional Size Of Software Maintainability Requirements. Journal of Theoretical & Applied Information Technology, 96(12).
[S4] Lin, C. J., & Yeh, D. M. (2016, December). A Software Maintenance Project Size Estimation Tool Based On Cosmic Full Function Point. In 2016 International Computer Symposium (ICS) (pp. 555-560). IEEE.
[S5] Ebert, C., & Soubra, H. (2014). Functional size estimation technologies for software maintenance. IEEE Software, 31(6), 24-29.
[S6] Kula, R. G., Fushida, K., Yoshida, N., & Iida, H. (2013). Micro process analysis of maintenance effort: an open source software case study using metrics based on program slicing. Journal of Software: Evolution and Process, 25(9), 935-955.
[S7] Elish, M. O. (2013). An exploratory study of package metrics as change size indicators in evolving object-oriented software. Computer systems science and engineering, 28(4), 251-257.
[S8] Wirotyakun, A., & Netisopakul, P. (2012, May). Improving software maintenance size metrics A case study: Automated report generation system for particle monitoring in Hard Disk Drive Industry. In 2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE) (pp. 334-339). IEEE.
[S9] Chua, B. B., & Verner, J. (2011, December). Evaluating Software Maintenance Effort: The COME Matrix. In International Conference on Advanced Software Engineering and Its Applications (pp. 120-136). Springer, Berlin, Heidelberg.
[S10] Nishizono, K., Morisakl, S., Vivanco, R., & Matsumoto, K. (2011, September). Source code comprehension strategies and metrics to predict comprehension effort in software maintenance and evolution tasks-an empirical study with industry practitioners. In 2011 27th IEEE International Conference on Software Maintenance (ICSM) (pp. 473-481). IEEE.

[S11] Durisic, D., Staron, M., & Nilsson, M. (2011, June). Measuring the size of changes in automotive software systems and their impact on product quality. In Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement (pp. 10-13).

[S12] Nguyen, V., Boehm, B., & Danphitsanuphan, P. (2011). A controlled experiment in assessing and estimating software maintenance tasks. Information and software technology, 53(6), 682-691.

[S13] Nguyen, V. (2010, September). Improved size and effort estimation models for software maintenance. In 2010 IEEE International Conference on Software Maintenance (pp. 1-2). IEEE.

[S14] Adamo, D. A., Fabrizi, S., & Vergati, M. G. (2007, March). A light functional dimension estimation model for software maintenance. In 2007 IEEE International Conference on Exploring Quantifiable IT Yields (pp. 73-78). IEEE.

[S15] Ahn, Y., Suh, J., Kim, S., & Kim, H. (2003). The software maintenance project effort estimation model based on function points. Journal of Software maintenance and evolution: Research and practice, 15(2), 71-85.

[S16] De Lucia, A., Pompella, E., & Stefanucci, S. (2002, July). Effort estimation for corrective software maintenance. In Proceedings of the 14th international conference on Software engineering and knowledge engineering (pp. 409-416).

[S17] Abran, A., Silva, I., & Primera, L. (2002). Field studies using functional size measurement in building estimation models for software maintenance. Journal of Software Maintenance and Evolution: Research and Practice, 14(1), 31-64.

[S18] Asl, M. H., & Kama, N. (2013, June). A change impact size estimation approach during the software development. In 2013 22nd Australian software engineering conference (pp. 68-77). IEEE.

[S19] Choudhari, J., & Suman, U. (2012). Story points based effort estimation model for software maintenance. Procedia Technology, 4, 761-765.

[S20] Hira, A., & Boehm, B. (2016, September). Using Software Non-Functional Assessment Process to Complement Function Points for Software Maintenance. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1-6).

[S21] Haoues, M., Sellami, A., & Ben-Abdallah, H. (2017). Functional change impact analysis in use cases: An approach based on COSMIC functional size measurement. Science of Computer Programming, 135, 88-104.

[S22] Sneed, H. M., & Huang, S. (2007, March). Sizing maintenance tasks for web applications. In 11th European Conference on Software Maintenance and Reengineering (CSMR'07) (pp. 171-180). IEEE.

[S23] Tran-Cao, D., & Levesque, G. (2003, September). Maintenance effort and cost estimation using software functional sizes. In International Workshop on Software Measurement, Montreal, Canada.

[S24] Kumar, S., Chakraverti, S., Agarwal, S. C., & Chakraverti, A. K. (2012). Modified COCOMO Model for Maintenance Cost Estimation of Real Time System Software. International Journal of Research In Engineering & Applied Sciences, 2(3).

[S25] Fioravanti, F., & Nesi, P. (2001). Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. IEEE Transactions on software engineering, 27(12), 1062-1084.