The Journal of Systems and Software 127 (2017) 52–77

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

A mapping study on design-time quality attributes and metrics

Elvira Maria Arvanitou[a], Apostolos Ampatzoglou[a,*], Alexander Chatzigeorgiou[b], Matthias Galster[c], Paris Avgeriou[a]

[a] Department of Mathematics and Computer Science, University of Groningen, The Netherlands
[b] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
[c] Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand

ARTICLE INFO

Article history:
Received 28 April 2016
Revised 20 December 2016
Accepted 25 January 2017
Available online 27 January 2017

Keywords:
Software quality
Measurement
Design-time quality attributes
Mapping study

ABSTRACT

Developing a plan for monitoring software quality is a non-trivial task, in the sense that it requires: (a) the selection of relevant quality attributes, based on application domain and development phase, and (b) the selection of appropriate metrics to quantify quality attributes. The metrics selection process is further complicated due to the availability of various metrics for each quality attribute, and the constraints that impact metric selection (e.g., development phase, metric validity, and available tools). In this paper, we shed light on the state-of-research of design-time quality attributes by conducting a mapping study. We have identified 154 papers that have been included as primary studies. The study led to the following outcomes: (a) low-level quality attributes (e.g., cohesion, coupling, etc.) are more frequently studied than high-level ones (e.g., maintainability, reusability, etc.), (b) maintainability is the most frequently examined high-level quality attribute, regardless of the application domain or the development phase, (c) assessment of quality attributes is usually performed by a single metric, rather than a combination of multiple metrics, and (d) metrics are mostly validated in an empirical setting. These outcomes are interpreted and discussed based on related work, offering useful implications to both researchers and practitioners.

1. Introduction

Software quality is an ambiguous term, in the sense that: (a) from the viewpoint of the user, quality is about how software meets its purpose, (b) from the developers' point of view, quality is about the conformance of software to its specifications, (c) from the product view, quality deals with the structural characteristics of the software, and (d) from a monetary viewpoint, quality is about the amount of money that a client is willing to pay to obtain it (Kitchenham and Pfleeger, 1996). Additionally, quality assurance cannot be performed in the same way across different software projects. Instead, assuring the levels of quality for a specific project requires answering the following questions, as outlined in Fig. 1:

- ***What quality attributes should be monitored***? One of the first activities in software development is the selection of quality attributes (QAs) that are the most important for the specific project (usually termed as forces or architecture key-drivers) (Bass et al., 2003). Quality attributes are project-specific since different software applications have different priorities, concerns and constraints. Nevertheless, we anticipate that projects belonging to the same application domain are presenting a similar prioritization for their key-drivers (Eckhardt et al., 2017). For example, critical-embedded systems put special emphasis on run-time quality attributes (e.g., performance, energy efficiency, etc.), whereas applications with intense interaction with the users (e.g., enterprise applications), focus on design-time ones (e.g., maintainability, extendibility, etc.). However, monitoring quality attributes cannot be performed in the same way in all phases of software development, in the sense that different phases focus on different quality aspects of the software. For example, during the requirements phase the engineers are expected to be less focused to code-level quality aspects (e.g., cohesion, coupling, etc.), whereas during the testing phase the engineers are more probably concerned about the correctness and completeness of the implementation. Therefore, quality attributes should not only be prioritized by application domain, but by development phase, as well.
- ***How can these quality attributes be monitored?*** After selecting the quality attributes of interest for every type of product development phase, the next step is the development of a measurement plan to monitor the levels of the specific quality attributes, given the constraints of the

* Corresponding author.
E-mail addresses: e.m.arvanitou@rug.nl (E.M. Arvanitou), a.ampatzoglou@rug.nl, apostolos.ampatzoglou@gmail.com (A. Ampatzoglou), achat@uom.gr (A. Chatzigeorgiou), mgalster@ieee.org (M. Galster), paris@cs.rug.nl (P. Avgeriou).

http://dx.doi.org/10.1016/j.jss.2017.01.026
0164-1212/

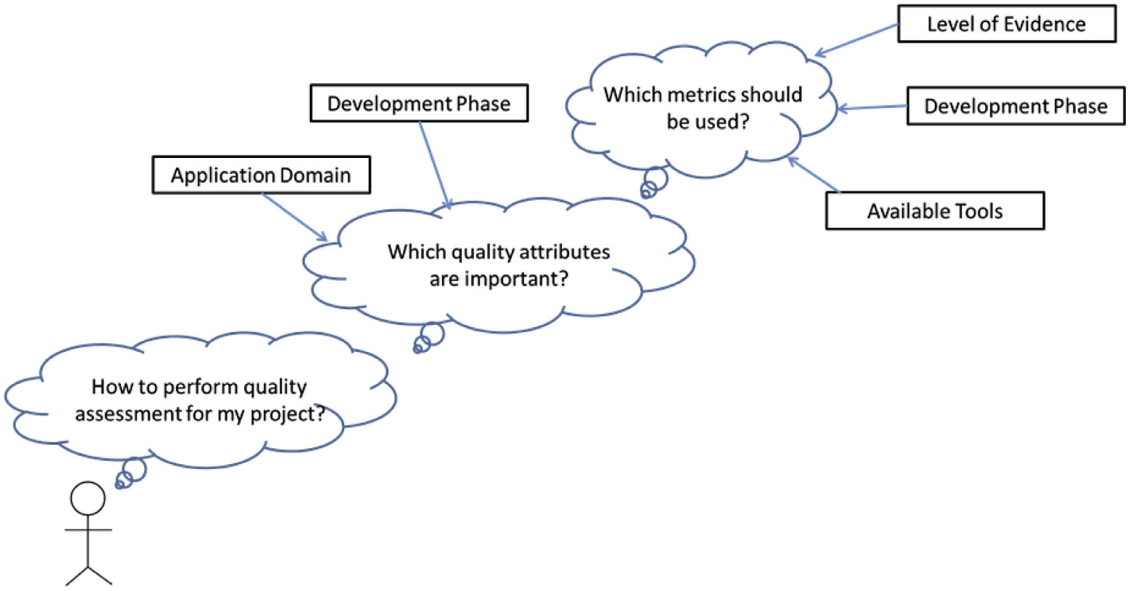

Fig. 1. Motivation of the study.

specific phase (e.g., available artifacts) (ISO/IEC 25023, 2003). However, there are no widely accepted sets of metrics for assessing a quality attribute across all development phases, since: (a) there is no set of metrics that is appropriate for all phases, and (b) quality attributes are not always associated with metrics. The usefulness of metrics that are accurately mapped to quality attributes has been extensively discussed by Harrison et al. (1998). However, only lately there have been efforts to develop a quality model where quality attributes are associated with measurable elements. For example, ISO/IEC 25023 (2003) provides measures for the characteristics in the product quality model. An additional information that is needed for the selection of specific metrics is their validity as assessors of the targeted quality attribute, and the availability of tools that can automate their calculation.

The goal of this study is to provide the necessary guidance to researchers and practitioners for answering the aforementioned questions. In this paper, we summarize the state-of-research on design-time quality attributes and metrics by conducting a mapping study. Therefore, the goal of this paper is to conduct a fair overview of "good quality" studies[1] on design-time quality attributes and related quality metrics. In particular, we identify and analyse research on quality in software engineering, without focusing on any programming paradigm / language, any application domain (e.g., telecommunication, embedded systems), or any software engineering phase (e.g., requirements engineering, software architecture, etc.). Thus, the outcome of this study provides the following contributions:

[c1] ***Highlight the most important design-time quality attributes in different application domains and development phases.*** This overview contributes a comprehensive list of design-time quality attributes that have been identified in different application domains, and which are of paramount importance in each development phase. Based on this, researchers can spot: (a) the most important design time quality attributes for each domain and development phase, and propose domain- or phase-specific approaches that tackle them, and (b) the aspects of quality in a specific application domains or development phases that have not been studied in detail and therefore might require more attention. Practitioners can use this comprehensive list of design time quality attributes as a checklist to find potential quality attributes for their particular project in every phase of development, based on the application domain in which the project belongs. Based on the outcome of this contribution, practitioners will be able to perform the process for quality attribute selection.

[c2] ***A mapping of design-time quality attributes and metrics***. Software metrics are used to quantify quality attributes. Thus, our study compiles a catalogue of metrics related to design-time quality attributes. In particular, we study five perspectives of this relation:

(a) we identify if a quality attribute is quantified through a formula that is based on aggregating other metrics, or is assessed through a set of metrics that cannot be aggregated (i.e., the quality attribute would be measured by individual metrics),
(b) we map quality attributes to the metrics that can be used for their quantification,
(c) we present the validation on the relationship between metrics and quality attributes and the provided level of evidence,
(c) we discuss the development phase in which different metrics can be calculated, and
(d) we provide a list of tools that can be used for automatically calculating the metric scores for a specific system.

By exploiting these five perspectives, practitioners can be guided in their metric selection and application processes. More specifically, after a practitioner picks a quality attribute for each development phase (based on ***c1***), he/she: (i) inspects how the quality attribute can be quantified (through a formula or a set of metrics), (ii) after checking the available metrics for its quantification in the current phase, and considering their validity levels, he/she can select the set of metrics that will be used, and (iii) based on the selected metrics, he/she will decide which tools can be reused or developed from scratch. Similarly, researchers can check which quality attributes are well-supported by metrics and

[1] The term "good quality" studies is used in this paper, as introduced by Kitchenham et al. (2009a). Based on their study "poor quality studies" are more likely to be identified in broad searches that are not targeting specific, established venues.

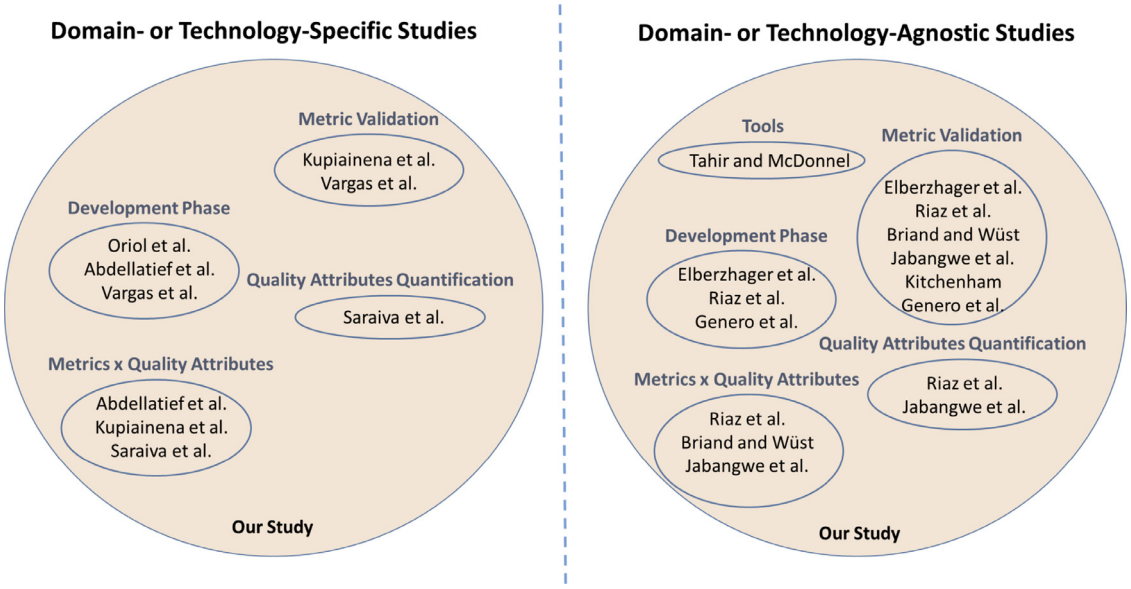

Fig. 2. Comparison to related work.

which quality attributes might require novel metrics. Additionally, based on metric validity assessment, researchers can identify quality attributes whose quantification requires further evaluation.

In this mapping study, we are interested only in "good quality" studies, in order to provide researchers and practitioners with an analysis of thoroughly conducted, validated and reliable research efforts (for further justification on the selection of "good quality" studies, see Section 3.2). Therefore, we aimed at identifying studies published only in particular "good quality" venues (more details on venue selection are presented in Section 3.2). Additionally, we focus on studies that *introduce* or *evaluate* quality attributes and metrics, excluding papers that *use* metrics for other software engineering purposes. We exclude these studies, since we expect that the employed metrics have already been identified in the studies in which they have been introduced (if published in one of the examined publication venues). Using snowballing to identify metric definitions that were published at venues outside the searching scope of the study was not applied since searching specific publication venues already resulted in a large number of primary studies.[2] Nevertheless, in such papers metrics are only the means for conducting the study rather than the goal/focus of the study. Therefore, they have been excluded from our secondary study. Including them as primary studies, would bias the obtained dataset by double-counting metrics that are used for different reasons. For example, a study that uses Coupling Between Objects (CBO) and Lack of Cohesion of Methods (LCOM) (Chidamber and Kemerer, 1994) to measure the effect of applying a certain refactoring is not evaluating the usefulness of the metrics, but the usefulness of the refactoring. Furthermore, since there is a lot of literature on software quality (Jabangwe et al., 2004), we decided to narrow the scope of this study to one type of quality attributes, namely *design-time quality attributes* (Abran and Moore, 2004). The Software Engineering Body of Knowledge (SWEBOK) defines design-time quality attributes as any aspect of software quality that is not discernible at run-time, (e.g., modifiability, reusability) (Abran and Moore, 2004).

[2] In total, we have examined more than 2800 articles, and therefore, we are confident that we have included the majority of "good quality" studies, published in the selected venues. Further increasing the number of primary studies would seriously threaten the feasibility of this work and introduce additional threats to validity, e.g., due to additional filtering of articles.

The rest of the paper is organized as follows: In Section 2, we discuss other secondary studies that are related to quality attributes or metrics. Next, in Section 3, we present the systematic mapping protocol, whereas in Sections 4 and 5, we present and discuss the results of this mapping study, respectively. Finally, in Section 6, we present threats to validity, and in Section 7 we conclude the paper.

2. Related work

In this section, we present secondary studies (namely systematic literature reviews and mapping studies) that are related to quality attributes and metrics. Whenever possible, we compare the goals of related work to our study and discuss points of differentiation. We do not discuss secondary studies that focus on runtime quality attributes, e.g., fault prediction (Catal and Diri, 2009; Radjenović et al., 2013), reliability (Febrero et al., 2014), etc., since our work is exploring design-time qualities. The rest of the section is organized into two sections: (a) studies that are application domain- or technology-agnostic, and (b) studies that are application domain- or technology-specific. As technology-agnostic we characterize studies that do not aim at a specific programming paradigm or language. In Fig. 2, we summarize the relation of our study to state-of-the-research on the topic of quality attributes and metrics. A detailed comparison between our study and individual related work is provided in Sections 2.1 and 2.2.

2.1. Domain- or technology-agnostic studies

Tahir and MacDonell (2012) published a mapping study on *dynamic metrics and software quality*. Their work identified dynamic metrics (i.e., metrics that capture the dynamic behavior of the software) that: (a) have been most frequently studied, and (b) could be recommended as topics for future research. To achieve this goal, they searched for articles in a list of eight journals and nine conferences/workshops. Sixty studies were identified and evaluated. As a result, they extracted a strong body of research associated with dynamic coupling and cohesion metrics, with most articles also addressing the abstract notion of software complexity. In a similar context, Elberzhager et al. (2012) presented a mapping study on the combination of static and dynamic quality assurance techniques (e.g., reported effects, characteristics, and constraints). The

search was based on four digital libraries (Inspec, Compendex, IEEE Xplore, and ACM DL). Fifty-one studies were selected and classified. The results suggest that the combination of static and dynamic analysis is an interesting research topic for enhancing code inspection and testing activities. The main point of differentiation of these studies, compared to our work is that we are interested in all types of software metrics, and not limited only on dynamic metrics.

Kitchenham (2010) conducted a preliminary mapping study to identify trends in influential papers on *software metrics*. The goal of this paper was to investigate: (a) the relationship between the influence of a paper and its publication venue (journal or conference) and (b) the type of validation performed on software metrics. To identify such papers, the author used Scopus and found: (a) the most cited papers in the years 2000–2005, and (b) the least cited papers in 2005. In particular, 87 papers were retrieved from IEEE and ACM DLs and Elsevier publications. The results suggested that the most cited papers were more frequently published in journals, and that empirical validation was the most popular type of metric evaluation rather than a theoretical one. Although this study partially overlaps with contribution c2c (***evidence related to the mapping between attributes and metrics***) the results are not directly comparable: Kitchenham (2010) included in her study papers related to the use of software metrics for particular types of software development activities (e.g. re-engineering or fault prediction), whereas our study is focused on papers that introduce and evaluate metrics.

Riaz et al. (2009) presented a systematic review on *software maintainability prediction and metrics*. Specifically, the study focused on finding models that are able to forecast software maintainability. In addition to that, they explored the level of evidence in these models and evaluate their significance. The search process was performed on 9 databases; however, all 14 papers have been retrieved only from 4 digital libraries. The results suggest that although the level of evidence on maintainability prediction is rather limited, the models of van Koten and Gray (2006), and Zhou and Xu (2008) are more accurate ones to predict maintainability. In a similar context, a recent mapping study by Jabangwe et al. (2004) reported evidence on the link between *object-oriented metrics and external quality attributes*. Jabangwe et al. focused on four quality attributes: reliability, maintainability, effectiveness, and functionality. To identify relevant studies, the authors queried five well-known digital libraries (ACM, IEEE, Scopus, Compendex, and Inspec) and identified 99 primary studies. Concerning design-time quality attributes, the most commonly studied one has proven to be maintainability, which in most of the cases is quantified through the Chidamber and Kemerer (CK) metric suite (1994). The results of the studies of Riaz et al. (2009), and Jabangwe et al. (2004) are comparable to ours; however, they both focus only on maintainability (at least in terms of design-time QAs).

Genero et al. (2005) and Briand and Wüst (2002) performed two literature surveys: (a) on *metrics that can be used on UML class diagrams*, and (b) on empirical studies that have been performed for evaluating *software quality models*. The main difference of these studies compared to ours is with respect to the employed methodology (i.e., survey versus a systematic mapping study). However, some of the results are comparable, since Genero et al. (2005) report on tools that have been proposed for quantifying metrics, and both studies (Genero et al., 2005; Briand and Wüst, 2002) discuss the level of empirical evidence related to well-known metric suites.

2.2. Domain- or technology-specific studies

Abdellatief et al. (2013) published a mapping study to investigate component-based software engineering (CBSE) metrics. In particular, the authors explored the granularity of metrics (system- or component-wide), the quality characteristics captured, and possible limitations of the state of the art. The search was performed on the following databases: ACM Digital Library, IEEE Explore, Springer Link, Scopus, ScienceDirect and Google Scholar. On the completion of the search process, 36 papers were selected. The results of the mapping study suggested that 17 of the proposed metrics can be applied to evaluate component-based software systems, while 14 can be applied to evaluate individual components. In addition, the outcome of this mapping study highlighted that only a few of the proposed metrics are properly defined. Concerning the overlap of the work of Abdellatief et al. to our study, we have identified three major points of differentiation. The work of Abdellatief et al.: (a) is focused only on CBSE systems—ours is paradigm-agnostic, (b) is focused mostly on metrics—our work is equally focused on metrics and quality attributes, and (c) includes papers that use metrics for particular types of software development activities—ours is focused only on studies that introduce and evaluate metrics/QAs.

Vargas et al. (2014) presented a mapping study that was dedicated to Serious Games (SGs). Specifically, the study aimed to identify important quality attributes and possible gaps in the research state of the art that deserve future investigation. The search process was performed on 6 digital libraries until April of 2013 (Scopus, ScienceDirect, Wiley, IEEE, ACM, and Springer). After applying the selection criteria, 112 studies were identified and classified (QAs, research results/methods, software artifacts, application area). The results of the study suggested that SG effectiveness and offered pleasure are the key-QAs in this domain, and that quality assessment is in the majority of the cases performed based on the final product. The work of Vargas et al. (2014) is different from our study, since we performed a mapping study without any restriction on the application domain, without focusing on the relevant QAs, but further elaborate on metrics that quantify them.

Saraiva et al. (2012) published a mapping study that investigated which metrics can be used to measure Aspect-Oriented software maintainability. The search strategy identified papers until June 2011 and was conducted on four digital libraries (IEEE, ACM, Compendex and ScienceDirect). At the end of the selection process, 138 primary studies were selected. The results of the review recommended a catalogue that can guide researchers in selecting which metrics are suitable for their studies. The work of Saraiva et al. (2012) presents substantial differences compared to our study, in the sense that Saraiva et al. focus on a specific programming paradigm (i.e., AOP) and a specific quality attribute (i.e., maintainability). Oriol et al. (2014) presented a mapping study to investigate quality models for web services. The goal of the study was to identify the: (a) quality models relevant to web services, (b) quality attributes that are referenced in the quality models, (c) definitions of the aforementioned quality attributes, and (d) most frequently investigated quality attribute across quality models. To achieve this goal, Oriol et al. (2014) searched 3 databases (Web of Science, IEEE and ACM) and retrieved 65 studies. The results of the study included 47 models for web services that in most of the cases include reliability, security and performance as quality attributes. Concerning the definition of quality attributes, Oriol et al. suggest that only 51% of the examined models have a unique and consistent definition for all their quality attributes. The major differences of our study to Oriol et al. (2014) are: (a) the domain specificity, and (b) metrics were outside the scope of Oriol et al. (2014).

Kupiainen et al. (2015) published a literature review on using metrics in industrial agile development. The goals of the study were to identify metrics, the reasons for applying them in an agile context, and the most influential metrics in industry. The search process was performed on: (a) Scopus and (b) the XP Conference 2013 proceeding because it could not be found through Scopus. Af-

Table 1
Overview of research state-of-the-art.

Reference		Demographics		Selection of QAs		Selection of Metrics			
	Year	Focus of study	#studies	Application domain	Dev. phases	Dev. phases	Validation	Tools	Mapping to QAs
Tahir and MacDonell	2012	Dynamic metrics	60					√	
Elberzhager et al.	2012	Metrics	51			√	√		
Kitchenham	2010	Metrics	87				√		
Riaz et al.	2009	Metrics maintainability	14			√	√		√
Jabangwe et al.	2014	Metrics reliability maintainability effectiveness functionality	99				√		√
Genero et al.	2005	Metrics design phase	13			√	√		
Briand and Wüst	2002	Quality models metrics	35				√		√
Abdellatief et al.	2013	Metrics CBSE	36			√			√
Vargas et al.	2014	QAs serious games	112	√					
Saraiva et al.	2012	Metrics AOP maintainability	138		√				√
Oriol et al.	2014	Quality models web services	65	√					
Kupiainen et al.	2015	Metrics agile	30			√			
Our study	2016	Metrics design-time QAs	154	√	√	√	√	√	√

ter applying their selection criteria, 30 studies were identified. The results of the study highlighted that the majority of agile metrics are related to the process (e.g., progress tracking, sprint planning, etc.). Additionally, Kupiainen et al. (2015) suggested that the most influential metrics in agile software development are velocity and effort estimates. The results of Kupiainen et al. (2015) focus on the process level, which is an important differentiation aspect, compared to our study, which is mostly interested in product metrics.

2.3. Overview

In Table 1, we provide an overview of related work and a comparison between our study and other secondary studies. The table is organized based on the expected contributions of our study: (a) important QAs for application domains and development phases, and (b) properties that can be used in metrics selection. Additionally, some demographics are reported (e.g., authors, year, etc.). We also present the primary focus of each study, in terms of: (a) QAs/metrics/both, (b) application domain, and (c) software development technology.

Based on the aforementioned table, we can highlight that our study goes beyond the state-of-research in various ways, as outlined below:

- Our study is the only one that investigates ***both metrics and quality attributes***. This point of differentiation is very important in the sense that based on such data we can provide a synthesis of results on both the metric and the quality attributes level.
- Our study is the ***largest*** one in terms of ***primary studies***, even though our searching space is limited to top venues only
- Our study is the ***broader*** one since it does not focus on specific application domains, development phases, or software development technologies. However, its level of detail does not lack depth compared to existing domain- or technology-specific studies, since it reports domain- and technology-specific results.

3. Study design

This section presents the protocol of the systematic mapping study. A protocol constitutes a plan that describes research questions and how the secondary study has been conducted. Our protocol is presented according to the guidelines of Petersen et al. (2008).

3.1. Objectives and research questions

The goal of this study, stated using the Goal-Question-Metrics (GQM) format (Basili et al., 1994) is: ***analyze*** existing literature on design-time quality attributes and related metrics for the ***purpose*** of characterization ***with respect to*** their: (a) popularity in the research community, (b) differences across application domains, development phases and programming paradigms, (c) empirical validation, and (d) tool support ***from the point of view of*** researchers and practitioners ***in the context of*** software quality assessment. Based on the aforementioned goal, we have set the following research questions:

RQ_1: Which quality attributes should be considered in a software development project, based on the application domain of the project and the current development phase?
- **$RQ_{1.1}$:** Which are the most studied quality attributes for each application domain?
- **$RQ_{1.2}$:** Which are the most studied quality attributes for each development phase?

RQ_1 is related to the selection of important quality attributes for a specific project (see contribution c1). $RQ_{1.1}$ and $RQ_{1.2}$ are expected to highlight differences in how quality is treated, according to the various backgrounds and needs of software engineers focusing on specific domains or development phases.

RQ_2: How can we effectively use quality metrics for assessing/quantifying a specific quality attribute?
- **$RQ_{2.1}$:** Can a quality attribute be quantified as a function of metrics?
- **$RQ_{2.2}$:** Which quality metrics are mapped to each quality attribute?
- **$RQ_{2.3}$:** How much evidence exists about the validity of quality metrics?
- **$RQ_{2.4}$:** What software quality metrics can be calculated in each development phase?
- **$RQ_{2.5}$:** Is there tool support for automatically calculating software quality metrics?

RQ_2 is related to contribution c2. $RQ_{2.1}$ is considered important since the quantification of the levels of quality attributes is needed for the objective assessment of the quality attributes. $RQ_{2.2}$ is auxiliary to $RQ_{2.1}$ since it provides a mapping between attributes and the particular metrics used to assess them. We note that the difference between $RQ_{2.1}$ and $RQ_{2.2}$ is that $RQ_{2.1}$ only focuses on which

QAs can be assessed with metrics, and not through which metrics they can be assessed. On the contrary, $RQ_{2.2}$ focuses exactly on which metrics can be used for quantifying certain QAs. To enhance the readability of this manuscript, in this paper we present metrics only related to the most frequent quality attributes, but the rest are still available in the accompanying technical report.[3] Additionally, $RQ_{2.3}$ highlights which metrics have been validated theoretically, empirically, or in both ways, and therefore are safer to be used (Briand et al., 1999). We ask $RQ_{2.4}$ for similar reasons as $RQ_{1.2}$, i.e., to investigate which quality metrics are applicable in every development phase, and which are the most popular ones in each phase. Finally, $RQ_{2.5}$ aims at recording the tools that can be used for automating the calculation of metrics, thus supporting the quality assessment process.

3.2. Search process

We defined our search strategy considering the goal and research questions of the study. Specifically, we have selected not to perform a search of the complete content of digital libraries, but to take into account only a limited number of selected venues. As explained in the introductory section, "quality" is a broad and often ill-defined concept: A vast portion of software engineering literature touches on quality, since the ultimate goal of software engineering research and practice is to ensure or improve the quality of software systems. Consequently, we focus our search on "good quality studies", i.e., high-quality papers published at premium software engineering venues. As described by Kitchenham et al., targeted searches at carefully selected venues are justified to omit low quality papers (Kitchenham et al., 2009a) and to avoid low quality grey literature (Kitchenham et al., 2009a). The proposed search approach, i.e., selecting specific publication venues has been applied in other systematic secondary studies in the field of software engineering, such as (Galster et al., 2014; Cai and Card, 2008; Kitchenham et al., 2009b).

3.2.1. Selection of publication venues

Our search method is based on Cai and Card (2008), where the authors selected seven journals and seven conferences as the search space for their secondary study. In addition to selecting only top venues of software engineering research, we explore only general software engineering venues, and not venues related to software engineering phases (e.g., architecture, maintenance, validation and verification, etc.) or application domains (e.g., embedded systems, multimedia applications, etc.). The criteria that have been taken into account while selecting the publication venues where:

- **cr.1.** We only included venues which are classified "Computer Software" by the Australian Research Council and evaluated higher than or equal to level "B" (for journals) and "A" (for conferences). We included venues with "B" because rankings of scientific venues are usually not conclusive and vary between ranking systems. We consider "Computer Software", because it is the category that includes the publication venues related to software engineering (among other computer science disciplines that are included in "Computer Software").
- **cr.2.** Searched venues had to be strictly from the software engineering domain. The category "Computer Software" also contains venues that do not focus on software engineering. Other venues of very high quality and with a high ranking and a large field rating (such as Communications of the ACM) are excluded since they target a diverse audience and therefore typically do not present in-depth research studies on specific topics.
- **cr.3.** Searched venues should not be related to a specific software engineering application domain or development phase. Thus, venues of very high quality, with a high ranking and a large field rating (such as the International Requirements Engineering Conference) are excluded since they target specific domains/phases.
- **cr.4.** We used the field rating of venues provided by Microsoft Academic Research (http://academic.research.microsoft.com) as the final criterion for venue quality. More specifically, we exclude venues that do not have a field rating value. The field rating is similar to the h-index, since it calculates the number of publications by an author and the distribution of citations over publications. Field rating only calculates publications and citations within a specific field and shows the impact of the scholar or journal within that specific field. The field rating from Microsoft Academic Research is, to the best of our knowledge, the only source where you can extract the same venue quality measures for both journals and conferences. Other measures, such as impact factor or acceptance rates have not been taken into account, because they are not uniform across journals and conferences. Furthermore, impact factors and acceptance rates are not available from one common source for all venues but would need to be gathered from different sources, causing threats to the reliability of the study.

The list and the scoring of each venue with respect to the abovementioned criteria are presented in Appendix A, organized by criteria (cr.1 to cr.4). The results of Appendix A, in terms of journals are identical to those of Wong et al. (2011), who used the same seven journals for assessing top software engineering scholars and institutions (Wong et al., 2011). Concerning conferences, the results are in general in accordance to those of Cai and Card (2008), by taking into account that we have excluded conferences specific to development phases (ISSTA and ISSRE); thus we agree in the selection of four out of five conferences. The difference is on the substitution of the Annual Computer Software and Application Conference (COMPSAC) with the International Conference on Software Process (ICSP). COMPSAC is not rated from the Australian Research Council with an "A" ranking and therefore it was not included in the considered publication venue set.

The distribution channels that have been used for accessing the identified studies are the digital libraries, in which each venue is publishing their accepted articles. Therefore, we used the ***IEEE Digital Library*** for *Transactions on Software Engineering* (TSE), *International Conference on Software Engineering* (ICSE), *International Symposium on Empirical Software Engineering and Measurement* (ESEM), *International Conference* on *Automated Software Engineering* (ASE), *International Conference on Software Processes* (ICSP), and *IEEE Software* (SW). We used the ***ACM Digital Library*** for identifying studies published in *Transactions on Software Engineering and Methodology* (TOSEM) and *International Symposium on the Foundations of Software Engineering* (FSE). Articles published in the *Empirical Software Engineering* (ESE) journal have been retrieved from ***Springer***, whereas papers published in *Software: Practice and Experience* (SPE) have been accessed through the ***Wiley*** on-line library. Finally, primary studies published in *Information and Software Technology* (IST) and *Journal of Systems and Software* (JSS) have been retrieved from the ***ScienceDirect*** library.

3.2.2. Search string and search strategy

As keywords for the search string we have chosen to use simple and generic terms, which may yield as many meaningful results as possible without any bias or preference to a certain quality at-

[3] http://www.cs.rug.nl/search/uploads/Resources/QA-QM-TR-2016-04.pdf.

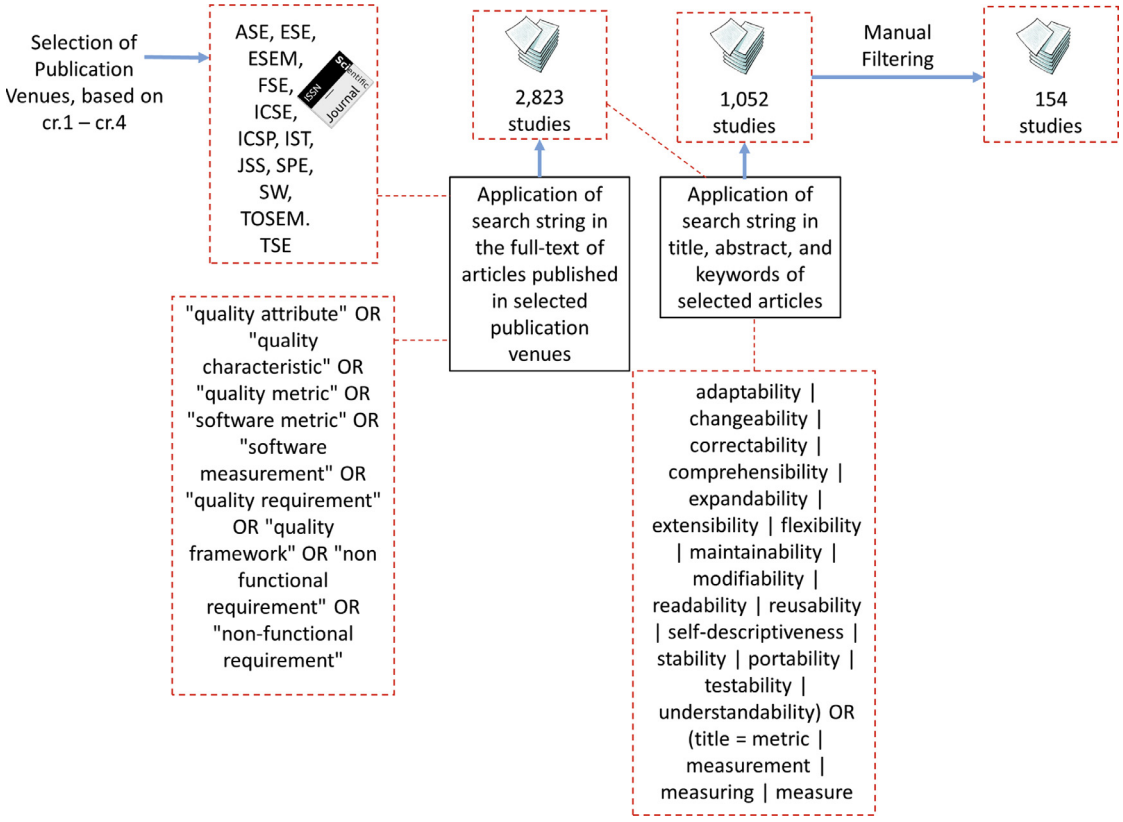

Fig. 3. Overview of search process.

tribute. The search string has been applied to the full text of the manuscripts of all selected venues, without any time constraints (we included articles published until June 2015). The search has been conducted automatically through the digital libraries of each venue. The final search string was:

> "quality attribute" OR "quality characteristic" OR "quality metric" OR "software metric" OR "software measurement" OR "quality requirement" OR "quality framework" OR "non-functional requirement" OR "non-functional requirement"

The search string was adjusted based on the capabilities of the search engines. We note that all terms have been used in their singular form, since their equivalent plural form would be identified through this search, in the sense that the singular form is a sub-string of the plural one. In addition to that the majority of search engines check plural automatically, and it is also expected that the singular form of a word would be used in at least one of the search fields (e.g., abstract, keywords, etc.). In order to validate the fitness of the used search string we have used a "quasi-gold" standard as proposed by Zhang et al. (2011). This "quasi-gold" standard has been defined by manually searching the issues or proceedings of all venues (see Section 3.2.1 and Appendix A) for relevant primary studies during 2014–2015. When manually searching the venues, we have considered the full text. All studies identified in the manual search were also found in the automatic search using the search string. This gives us confidence that the search string was accurate.

3.2.3. Overview of selection process

In our systematic mapping, the selection of candidate primary studies has been performed by automated search in specific publication venues (an overview is provided in Fig. 3).

In the primary selection procedure, the defined search string has been applied to each publication source listed in the table of Appendix A. As a result, a set of primary studies possibly related to the research questions has been obtained. After this step, Table of Contents (TOCs), editorials, keynotes, panels, workshop summaries, and biographies were manually removed from the candidate primary studies. The retrieved dataset was then automatically filtered based on the application of the following search string on each study's title, abstract and keyword:

> (title | keywords | abstract = adaptability | changeability | correctability | comprehensibility | expandability | extensibility | flexibility | maintainability | modifiability | readability | reusability | self-descriptiveness | stability | portability | testability | understandability) OR (title = metric | measurement | measuring | measure)

The design-time quality attributes used in the aforementioned filtering were retrieved from ISO/IEC/IEEE 24765:2010 vocabulary (2010). Based on this set, the full-text of each primary study was read and evaluated based on the inclusion and exclusion criteria (see Section 3.3). We note that we have selected to first search in the full-text of the articles in order to apply a uniform process in all selected digital libraries, since SpringerLink does not allow the application of our search string in title/abstract/keywords. After retrieving the first dataset (by querying the full-text of the manuscripts), the second filtering was performed on an external tool, namely JabRef.

3.3. Article filtering phases

Another important element of the systematic mapping planning is to define the Inclusion Criteria (IC) and Exclusion Criteria (EC).

These criteria are used as guidelines for including primary studies that are relevant to answer the research questions and exclude studies that do not help answer them. A primary study is included if it satisfies one or more ICs, and it is excluded if it satisfies one or more ECs. The inclusion criteria of our systematic mapping are:

- IC1: The primary study defines one or more design-time quality attributes;
- IC2: The primary study defines one or more design-time metrics;
- IC3: The primary study evaluates one or more design-time metrics;

This mapping study takes into account all typical exclusion criteria, e.g. the paper is written in a language other than English or was published as grey literature, which however are covered by our systematic selection of venues. The only other exclusion criterion established is:

- EC1: The primary study is an editorial, position paper, keynote, opinion, tutorial, poster or panel.
- EC2: The primary study uses metrics for evaluating a software engineering process, method, or tool, without evaluating the metric per se or its ability to assess a quality attribute.
- EC3: The primary study introduces or validates quality attributes / software metrics that concern run-time properties, or business / process indicators.

Every article selection phase has been handled by two members of the team and possible conflicts have been resolved by the other three members. For each selected publication venue, we have documented the number of papers that were returned from the search and the number of papers finally selected (see Section 4).

3.4. Keywording of abstracts (classification scheme)

In the study by Petersen et al. (2008), the authors propose keywording of abstracts as a way to develop a classification scheme for primary studies and to answer the research questions, if existing schemes do not fit, and ensure that the scheme takes into account the identified primary studies. In our study, we expected that from the majority of primary studies we would not be able to extract all information required to answer the research questions from abstracts. As a consequence, we decided to apply the keywording technique to the full text of the manuscripts, i.e., to read the full text of the studies in order to identify the values of the following variables (used as classification dimensions): (a) quality attributes, (b) quality metrics, and (c) software development phases.

3.5. Data collection

During the data collection phase, we collected a set of variables that describe each primary study. At this point it is necessary to clarify that some variables presented in the keywording process are included in this list as well to increase text uniformity and the completeness of the list. Data collection was handled by two members of the team and possible conflicts were resolved by the other researchers. For every study, we extracted and assigned values to the following variables:

[V1] Title: Records the *title* of the paper.
[V2] Author: Records the *list of authors* of the paper.
[V3] Year: Records the *publication year* of the paper.
[V4] Type of Paper: Records if the paper is published in a *conference* or *journal*.
[V5] Publication Venue: Records the *name* of the corresponding journal or conference.
[V6] Application Domain: Records *generic* if the results are domain-agnostic, or the *name of the specific domain*, if the results are domain-specific. For example, as application domains we have used: embedded systems, enterprise applications, web applications, etc.
[V7] Development Phase: Records the development phase that is investigated in the primary study (e.g., *requirements, architecture, design, implementation, testing*)
[V8] Relationship of Study with Quality: Records if the study *introduces* or *evaluates* a quality attribute or metric. We note that studies only *using* a quality attribute or a metric suite (without validating them) have been excluded in the article selection phase.
[V9] Names of Quality Attributes: Records *a list of the names of quality attributes* investigated in the study. We note that QAs should be explicitly mentioned in the paper, and are recorded with the exact name introduced in the paper.
[V10] Quality Attribute Associated with Quality Metric: Records *yes*, if the study is associating a quality attribute with quality metrics (e.g., CBO is connected with maintainability (Li and Henry, 1993), or *no* if not.
[V11] Quantification of Quality Attributes through Quality Metrics: Records *yes*, if the association of a quality attribute to a set of metrics is able to produce a score for the QA (e.g., reusability $= 0.5 * $ Class Interface Size $+ 0.5 *$ Design Size in Classes $+ 0.25 *$ Cohesion Among Methods of a Class – $0.25 *$ Direct Class Coupling (Bansiya and Davies, 2002), or *no*, if not.
[V12] Names of Quality Metrics: Records *a list of the names of quality metrics* investigated in the study.
[V13] Programming Paradigm: Records the programming paradigm in which a quality metric can be calculated (e.g. *object-oriented, functional*)
[V14] Tool Availability for Quality Metrics: Records if a metric/metric suite can be automatically calculated by a tool. In particular, we record *created by author,* if the authors created a tool for this reason, but without assigning a name. If the authors reuse a tool or assigned a name to the tool that they have created, we record the *tool name*.
[V15] Quality Metric Level of Evidence: Records the level of metric validation. According to Briand et al. (1999), metric validation should be performed as a two-step process, i.e., providing a theoretical and empirical validation. The theoretical validation aims at mathematically proving that a metric satisfies certain criteria, whereas the empirical one aims at investigating the accuracy with which a specific metric quantifies the corresponding quality attribute. Therefore, if a study validates a metric only in a theoretical manner, we record *theoretical*. If a study performs, only an empirical validation, we record *its empirical validation ranking*, using the six levels of evidence as described by Alves et al. (2010):
  1. No evidence.
  2. Evidence obtained from demonstration or working out toy examples.
  3. Evidence obtained from expert opinions or observations.
  4. Evidence obtained from academic studies (e.g., controlled lab experiments).
  5. Evidence obtained from industrial studies (e.g., causal case studies).
  6. Evidence obtained from industrial evidence.

Finally, if a study validates a metric both ways, we record *complete validation*.

We note that some fields have been marked as “N/A” for some primary studies. For example, if a primary study defines QAs without referencing ways of measuring them, variables [V10]–[V15]

Table 2
Mapping of paper attributes to RQs.

Research question	Variables used	Synthesis method
$RQ_{1.1}$	[V6], [V9]	Crosstabs for [V6], [V9]
$RQ_{1.2}$	[V7], [V9]	Crosstabs for [V7], [V9]
$RQ_{2.1}$	[V9], [V10], [V11]	Crosstabs for [V9], [V10] Crosstabs for [V9], [V11]
$RQ_{2.2}$	[V12], [V9]	Crosstabs for [V12], [V9]
$RQ_{2.3}$	[V12], [V15]	Crosstabs for [V12], [V15]
$RQ_{2.4}$	[V7], [V12]	Crosstabs for [V7], [V12]
$RQ_{2.5}$	[V12], [V14]	Crosstabs for [V12], [V14]

Table 3
Study selection per publication venue.

Source	Papers returned	Papers automatically filtered by title/abstract	Papers included
ASE	47	18	2
ESE	229	72	11
ESEM	172	64	9
FSE/ESEC	19	8	1
ICSE	165	61	7
ICSP	8	2	0
IST	674	240	37
JSS	800	326	47
SPE	136	40	4
SW	270	69	3
TOSEM	13	5	3
TSE	290	147	30
Total	**2823**	**1052**	**154**

have been left blank. Similarly, if one primary study uses only software quality metrics, without defining the QA that they quantify, variables [V9] – [V11] have been left blank.

3.6. Data analysis

Variables [V1] – [V5] are used for documentation reasons. The mapping between the rest of the variables and research questions is provided in Table 2, accompanied by the synthesis or analysis methods used on the data. For both research questions, demographics have been provided through a frequency table of variables [V9] and [V12], respectively for RQ_1 and RQ_2.

4. Results

After searching and filtering as described in Sections 3.2 and 3.3, we obtain a dataset of 154 primary studies. In Table 3, for each considered publication venue, we present the number of papers that have been returned as candidate primary studies (step 1), the number of papers qualified after primary study selection based on title and abstract (step 2), and the final number of primary studies (step 3)—see Fig. 3.

The main reason for excluding papers in the manual filtering phase (based on the full-text inspection) was that the studies did not concern design-time quality attributes but either business-related attributes (e.g., cost / effort / productivity estimation) or run-time quality attributes (e.g. reliability, safety, fault prediction and defect proneness). In the rest of this section, we present the results of our mapping study, organized based on the research questions. Therefore, in Section 4.1 (and its sub-sections) we discuss our findings related to the quality attributes, whereas in Section 4.2, we present our findings concerning software quality metrics.

4.1. Design-time quality attributes

As a starting point for our investigation, in Table 4, we list the frequencies of design-time quality attributes as they appear in primary studies. In this table, apart from the name of the quality attribute and the number of studies in which it has been investigated, we also present its usual level in hierarchical quality models (e.g., QMOOD (Bansiya and Davies, 2002), ISO-9126 (2001)). In particular, as Low-Level (LL) we characterize only quality attributes of the lowest level, i.e., those that can be directly calculated from metrics (e.g., coupling is calculated through CBO). High-Level (HL) quality attributes are attributes at any other level (e.g., maintainability, stability, etc.[4]). To keep the table relatively short, we list only the quality attributes that have been investigated in more than two studies.

From the results of Table 4 various observations can be made:

- First, most of the *low-level quality attributes* are concentrated in the *top-frequency positions*. In particular, 4 out of 6 LL_QAs are ranked in the top-12 QAs, with regards to their frequency. By performing a Spearman correlation between the level of QA (High/Low) with their frequency ranking, we have validated the existence of such a relationship with moderate strength (sig: 0.04, coefficient: 0.33). This is a rather intuitive result, in the sense that LL_QA's are easier to manage, due to their direct association to metrics. The most frequently occurring LL_QAs are ***complexity***, ***cohesion***, and ***coupling***, which are the backbone of object-oriented programming and many programming principles (Martin, 2003; Larman, 2004).
- Second, the number of *high-level quality attributes* is higher than the number of low-level ones (i.e., in the table we can identify 27 HL_QAs and 6 LL_QAs).
- Third, the top-studied HL_QA based on frequency is ***maintainability*** (approximately, 1 out of 5 studies), followed by ***understandability*** and ***stability***. Although understandability is partially related to maintainability, we preferred to separately report on its importance for two reasons: (a) some quality models treat them separately. For example, QMOOD (Bansiya and Davies, 2002) differentiates between flexibility, extensibility and understandability, and ISO-9126 (2001) describes understandability as a sub-characteristic of usability,[5] (b) software understandability has managed to create a different research community around it, as implied by the existence of many related, well-established conferences, e.g., the International Conference on Program Comprehension (ICPC).

4.1.1. Quality attributes and application domains ($RQ_{1.1}$)

In Table 5, we present the results of cross-tabulating HL_QAs and application domains. In particular, for every application domain, we record the name of HL_QAs and the number of studies in which they have been investigated. We note that this list is not exhaustive, since we only provide the most frequent quality attributes for each domain, whereas at the same time we excluded application domains that are discussed in only one study. Based on the obtained results, the majority of the studies (78%) introduced or validated a quality attribute without specifying the application domain. Regarding the rest of the studies, the most frequently studied application domains are: embedded systems (7 studies), information systems (5 studies), database applications (5 studies), web applications (4 studies), and distributed systems (3

[4] Based on ISO-9126 maintainability decomposes to analyzability, changeability, stability, testability, and compliance (2001).

[5] Although usability is intuitively perceived as run-time quality attribute, in this paper we only consider its design-time nature. For example, as a proxy of software usability one can investigate aspects like manual adequacy, functional size, etc.

Table 4
List of most frequently studied quality attributes.

Quality attributes	Level	Freq.	Quality attributes	Level	Freq.	Quality attributes	Level	Freq.
Complexity	LL	35	Change proneness	HL	7	Traceability	HL	4
Maintainability	HL	31	Size	LL	7	Documentation	HL	4
Cohesion	LL	29	Analyzability	HL	7	Portability	HL	4
Coupling	LL	17	Reusability	HL	7	Accuracy	HL	4
Stability	HL	15	Efficiency	HL	6	Operability	HL	4
Understandability	HL	15	Modularity	HL	6	Comprehensibility	HL	3
Testability	HL	15	Adaptability	HL	6	Correctness	HL	3
Functionality	HL	12	Learnability	HL	5	Abstraction	LL	3
Changeability	HL	11	Suitability	HL	5	Interoperability	HL	3
Usability	HL	11	Completeness	HL	5	Installability	HL	3
Inheritance	LL	9	Consistency	HL	5	Replaceability	HL	3
Modifiability	HL	9	Recoverability	HL	4	Effectiveness	HL	3

Table 5
List of most frequently QAs for each domain.

Application domain	Quality attributes	Freq.	Application domain	Quality attributes	Freq.
Generic	*Maintainability**	*23*	*Embedded systems*	*Correctness**	*2*
	*Stability**	*12*		*Maintainability**	*2*
	*Testability**	*11*		*Traceability*	*1*
	*Understandability**	*9*		*Completeness**	*1*
	*Changeability**	*8*		*Consistency**	*1*
	Change proneness	*7*		*Volatility*	*1*
	*Modifiability**	*7*		*Traceability*	*1*
	*Functionality**	*6*		*Completeness*	*1*
	*Usability**	*6*		*Consistency*	*1*
	Modularity	*5*		*Documentation*	*1*
	*Reusability**	*5*		*Memory Requirements*	*1*
	*Analyzability**	*3*		*Reliability*	*1*
	Efficiency	*3*		*Suitability*	*1*
	*Adaptability**	*3*	*Information systems*	*Functionality**	*2*
	*Completeness**	*3*		*Efficiency*	*2*
	Learnability	*2*		*Recoverability*	*2*
	Portability	*2*		*Maintainability**	*2*
	Documentation	*2*		*Traceability*	*2*
	*Comprehensibility**	*2*		*Understandability*	*2*
	*Consistency**	*2*		*27 additional QAs*	*1*
	*Effectiveness**	*2*	*Database application*	*Maintainability**	*2*
Web Applications	*Functionality**	*2*		10 additional QAs	*1*
	*Maintainability**	*1*	*Distributed systems*	*Maintainability*	*1*
	*Reusability**	*1*		*Comprehensibility*	*1*
				Locality	*1*
				Modifiability	*1*

* These terms are duplicate in the table.

studies). We note that for reporting the cross tabulation between application domains and quality attributes, we have preferred to use rather broad domains, so that results are meaningful. For example, we preferred to merge enterprise and business applications under the common term information systems, so that this broad domain to represented in 5 studies.

By comparing *domain-specific* and *domain-agnostic* studies, we can observe that ***maintainability***, ***testability***, ***understandability*** and ***changeability*** are the most frequently studied quality attributes, regardless of the application domain. Similarly, ***effectiveness***, ***comprehensibility***, ***documentation*** and ***portability*** are the least frequently studied domain-agnostic quality attributes. Concerning differences between *domain-specific* and *agnostic* studies, we have been able to identify two groups:

- *Quality attributes that are more frequently used in domain-specific studies*. In this group, we have classified ***functionality***, ***usability***, ***consistency***, ***learnability***, and ***analysability***. For example, functionality is of particular interest (50% of the studies) in application domains that are users-centric, e.g. enterprise, business, and web applications.
- *Quality attributes that are more frequently used in domain-agnostic studies*. In this group, we have classified ***stability***, ***change proneness***, ***modifiability***, and ***modularity***. An interesting observation concerning the quality attributes of this group is that all of them are sub-characteristics of maintainability. This is an indication that studies emphasizing on specific application domains are more probable to focus on maintainability, rather than others.

4.1.2. Quality attributes and development phases (RQ1.2)

In this section, we discuss the main findings of this study, with respect to the relationship between HL_QAs and development phases. In particular, in Table 6, we list the cross-tabulation of all software development phases with the most frequent HL_QAs.

From Table 6, we observe that during the ***maintenance*** phase, software engineering researchers are interested *only in maintainability and its sub-characteristics*. Similarly, during the ***design*** and ***implementation*** phases, the focus is again on *maintainability*, but with some additional emphasis on *testability*. In addition, *maintainability* is the most frequently studied QA, in all phases, except from ***project management*** and ***requirements*** engineering. Concerning the ***requirements*** phase, an interesting finding is the fact that 3 out of 5 most studied QAs (i.e., traceability, completeness, and consistency) are related only to the specific phase. Furthermore, the

Table 6
List of most frequently QAs for each development phase.

Development Phase	Quality attributes	Freq.	Development Phase	Quality attributes	Freq.
Implementation	*Maintainability**	*13*	*Project management*	*Functionality**	*4*
	*Stability**	*4*		*Usability**	*4*
	*Change proneness**	*4*		*Efficiency**	*3*
	*Modifiability**	*3*		*Maintainability**	*3*
	*Modularity**	*3*		*Analyzability**	*3*
	*Functionality**	*3*		*Changeability**	*3*
	*Testability**	*3*		*Stability**	*3*
				*Adaptability**	*3*
Design	*Maintainability**	*7*	*Architecture*	*Maintainability**	*7*
	*Testability**	*5*		*Functionality**	*3*
	*Understandability**	*5*		*Adaptability**	*3*
	*Modifiability**	*4*		*Efficiency**	*2*
	*Change proneness**	*3*		*Recoverability*	*2*
	*Analyzability**	*3*		*Usability**	*2*
	*Stability**	*3*		*Understandability**	*2*
				Learnability	*2*
				*Modularity**	*2*
Requirements	*Traceability*	*3*	*Maintenance*	*Maintainability**	*4*
	Completeness	*3*		*Changeability**	*3*
	Consistency	*3*		*Stability**	*1*
	*Stability**	*3*	*Testing*	*Testability**	*4*
	*Understandability**	*3*			

* These terms are duplicate in the table.

Table 7
List of most frequently studied quality metrics.

Quality metrics	LL_QAs	Frequencies
Lack of Cohesion of Methods-1 (LCOM1)	*Cohesion*	*23*
Lines of Code (LOC)	*Size*	*23*
Halstead n1	*Complexity*	*20*
Halstead n2	*Complexity*	*20*
Cyclomatic Complexity (CC-VG)	*Complexity*	*17*
Depth of Inheritance Tree (DIT)	*Inheritance*	*17*
Tight Class Cohesion (TCC)	*Cohesion*	*16*
Weighted Methods per Class (WMC)	*Complexity*	*15*
Response for Class (RFC)	*Coupling*	*15*
Number of Children (NOCC)	*Inheritance*	*14*
Coupling Between Objects (CBO)	*Coupling*	*13*
Number of Methods (NOM)	*Size*	*12*
Loose Class Cohesion(LCC)	*Cohesion*	*12*
Message Passing Coupling (MPC)	*Coupling*	*10*
Cohesion(Coh)	*Cohesion*	*10*
Lack of Cohesion of Methods-2 (LCOM2)	*Cohesion*	*10*
Lack of Cohesion of Methods-5 (LCOM5)	*Cohesion*	*10*
Data Abstraction Coupling (DAC)	*Coupling*	*9*
Lack of Cohesion of Methods-3 (LCOM3)	*Cohesion*	*9*
Lack of Cohesion of Methods-4 (LCOM4)	*Cohesion*	*8*

interplay between requirements and design that is addressed in the ***architecture*** phase is evident through the importance of *functionality* in the corresponding phase. Finally, as expected, *functionality* is more important during the first development phases, i.e., ***project management*** and architecting.

4.2. Quantification/Assessment of quality attributes through software metrics

In this section, we present the results of our mapping study related to software quality metrics. From the 154 primary studies originally obtained during our paper selection process, 136 papers (87.4%) involved software metrics. The rest of this section deals only with these studies. The section is organized by sub-research question. In Table 7, we present the top-20 most frequently studied metrics (answering RQ_2). In addition to that, in Table 7, we map the quality metric to the LL_QA that it quantifies.

Based on the results of Table 7, we can observe that all metrics proposed by ***Chidamber and Kemerer*** (1994) appear in the list of most frequently studied metrics. Concerning other metric suites, the only metric from the Li and Henry (1993) suite that is missing from Table 7 is SIZE2 (i.e., sum of attributes and methods). Additionally, in the list we can identify two metrics from the ***Halstead*** metric suite (Halstead, 1977) that are not specific for the object-oriented paradigm.

Regarding the *quantification of LL_QAs*, the results of Table 7 suggest that the most popular quality attribute that is supported by metrics is ***cohesion***, which is represented in the list with 8 metrics. A possible explanation for that is the debate on the accuracy and validity of LCOM1 and LCOM2, which opened a research direction on how cohesion should be quantified. Additionally, in Table 7 we can identify metrics that quantify both lack (e.g., LCOM1) and presence (TCC) of cohesion. Another interesting finding is that although complexity is the most studied low-level quality attribute (see Table 4) in Table 7 we can only identify 4 complexity metrics. This observation can be explained by the fact that all complexity metrics are very popular (4 out of top-8 metrics), implying that its quantification is rather uniform from researchers.

4.2.1. Quantification of quality attributes ($RQ_{2.1}$)

In this section, we discuss the extent to which quality attributes can be quantified through metrics. In particular, we discriminate between two categories: (a) QA that are assessed through a single metric, and (b) QAs that are assessed through the combination of more than one metrics. We clarify that this research question (i.e., *which quality attributes are more frequently associated with metrics*) is only relevant for HL_QAs, in the sense that the association between LL_QAs and metrics always exists (e.g., all cohesion metrics are related to cohesion) and quantifying LL_QA's can be mapped to metrics calculation (e.g., cyclomatic complexity = edges in a flow chart – nodes in a flow chart + 2*terminal nodes, CC = L – N + 2P (McCabe, 1976)). Also, we note that this RQ does not aim to associate specific metrics to quality attributes, but only investigates the existence of such a relationship. The exact mapping of metrics and quality attributes is provided by answering $RQ_{2.2}$.

The results gathered to answer $RQ_{2.1}$ are presented in Table 8. Specifically, for every HL_QA (row), we record the following information:

- ***Association to metrics***. The count of primary studies that associate the HL_QA to metrics (column: 2) and those that do not (column: 3); and
- ***Quantification through a combination of metrics***. Although a QA might be associated with some metrics (e.g., maintainability is related to CBO and LCOM), it is not always quantified as a function of these metrics. In these columns, we present the count of primary studies that provide a formula for quantifying the HL_QA based on metrics (column: 4), and those that do not (column: 5). We note that the count of columns 4 and 5, sums up to column 2, since studies that do not associate a QA with metrics are not able to provide such a formula.

From the results presented in Table 8, it can be observed that 66.1% of the studies associate HL_QAs with metrics. Furthermore, only ten HL_QAs have not been associated with any metric. However, this result does not imply that these QAs are not measurable, since they have been investigated in only one paper each. Therefore, it might be possible that metrics associated with these QAs might be presented in publication venues that have not been considered in this study. Additionally, approximately only half of HL_QAs (43.9%) use these metrics as factors in a function that provides an aggregated measurement for the quality attribute. The rest of the discussion, concerning $RQ_{2.1}$, is focused on the top-15 HL_QAs in terms of frequency (see Table 4):

- The HL_QA that is most *consistently related to metrics* is ***change proneness***, which has been associated with metrics in all 7 studies that it has been mentioned (100%).
- Similarly, high percentages can be observed for ***testability*** and ***reusability*** (86%), ***modularity*** (83%), and ***maintainability*** and ***stability*** (80%).
- The QA that has most frequently been associated with metrics, in absolute numbers, is maintainability, since 25 studies have associated at least one metric with it. The association of these QAs to metrics does not always lead to the *quantification of the attribute through a function that aggregates these metrics*.
- In particular, only 14% of studies provide a way to combine metrics to quantify ***change proneness***. This percentage is rather low also for ***modularity*** (16%), ***maintainability*** (29%), and ***stability*** (33%).
- The QAs that appear to be more frequently associated with metric aggregation functions are ***reusability*** (71%), ***functionality*** (58%), and ***testability*** (46%). We note that there is no study that quantifies modifiability through an aggregation function, despite the fact that 6 studies have connected it to several metrics.

Table 8
QAs quantification.

Quality attributes	QA associated with metrics		QA quantified as a formula of metrics	
	Yes	*No*	*Yes*	*No*
Maintainability	25	6	9	16
Stability	12	3	5	7
Understandability	9	6	6	3
Testability	13	2	7	6
Functionality	6	6	6	0
Changeability	7	4	3	4
Usability	6	5	3	3
Modifiability	6	3	0	6
Change Proneness	7	0	1	6
Analyzability	4	3	2	2
Reusability	6	1	5	1
Efficiency	2	4	2	0
Modularity	5	1	1	4
Adaptability	3	3	2	1
Learnability	*3*	*2*	*3*	*0*
Suitability	3	2	2	1
Completeness	4	1	1	3
Consistency	4	1	1	3
Recoverability	2	2	2	0
Traceability	2	2	0	2
Portability	1	3	1	0
Accuracy	2	2	2	0
Comprehensibility	3	0	1	2
Documentation	2	1	0	2
Correctness	2	1	0	2
Interoperability	*1*	*2*	*1*	*0*
Installability	*1*	*2*	*1*	*0*
Replaceability	*1*	*2*	*1*	*0*
Effectiveness	2	1	0	2
Readability	1	1	0	1
Locality	2	0	0	2
Volatility	2	0	0	2
Maturity	1	1	1	0
Co-existence	1	1	1	0
Conciseness	2	0	0	2
Clarity	1	0	0	1
Serviceability	1	0	0	1
Predictability	1	0	0	1
Compatibility	1	0	0	1
Unambiguity	1	0	0	1
Ambiguity	0	1	0	0
Compliance	1	0	1	0
Change impact	1	0	0	1
Evolveability	0	1	0	0
Sharing	0	1	0	0
Flexibility	0	1	0	0
Utilization	0	1	0	0
Openness	0	1	0	0
Soundness	1	0	0	1
Validity	1	0	0	1
Variability	1	0	1	0
Availability	0	1	0	0
Attractiveness	0	1	0	0
Customizability	0	1	0	0
Navigability	0	1	0	0
Simplicity	1	0	0	1
Total	**164**	**84**	**72**	**92**

4.2.2. Quality attributes and quality metrics ($RQ_{2.2}$)

In Table 9, we present the metrics that are more frequently used to assess high-level quality attributes. The list of high-level quality attributes in Table 9, is again not exhaustive, but is only limited to the top-10 most frequently studied high-level[6] quality attributes (see Table 4). We note that metrics that appear only in one study are omitted from the table (e.g., metrics for functionality, usability, changeability, and analyzability).

[6] In particular, from Table 4, we selected the top-11 quality attributes, since the 10th and the 11th quality attributes are having equal frequencies.

Table 9
Association between QAs and quality metrics.

Quality attributes	Quality metrics	Freq.	Quality attributes	Quality metrics	Freq.
Maintainability	*Depth of Inheritance Tree (DIT)* *	6	Change proneness	*Depth of Inheritance Tree (DIT)* *	3
	Lines of Code (LOC) *	6		*Number of Children (NOCC)* *	3
	Weighted Methods per Class (WMC) *	6		*Coupling Between Objects (CBO)* *	3
	Cyclomatic Complexity (CC-VG)	5		*Response for Class (RFC)* *	3
	Lack of Cohesion of Methods-1 (LCOM1) *	5		*Lack of Cohesion of Methods-1 (LCOM1)* *	3
	Tight Class Cohesion (TCC)	4		*Data Abstraction Coupling (DAC)* *	3
	Number of Children (NOCC) *	4		*Number of Attributes (NA)*	3
	Response for Class (RFC) *	4	Understand ability	*Lines of Code (LOC)* *	3
	Message Passing Coupling (MPC) *	4		*Depth of Inheritance Tree (DIT)* *	2
	Data Abstraction Coupling (DAC) *	4		*External Class Complexity (ECC)* *	2
	Number of Methods(NOM)	4		*External Class Size (ECS)* *	2
Reusability	*Lack of Cohesion of Methods-1 (LCOM1)**	3	Testability	*Response for Class (RFC)* *	4
	Lines of Code (LOC) *	2		*Coupling Between Objects (CBO)* *	3
	Coupling Between Objects (CBO) *	2		*Lack of Cohesion of Methods-1 (LCOM1)* *	3
	Response for Class (RFC) *	2		*Lines of Code (LOC)* *	3
	Message Passing Coupling (MPC) *	2	*Modifiability*	*Depth of Inheritance Tree (DIT)* *	2
	Weighted Methods per Class (WMC) *	2		*Halstead n1*	2
	Number of Children (NOCC) *	2		*Halstead n2*	2
	External Class Complexity (ECC) *	2	Stability	*Weighted Methods per Class (WMC)* *	2
	External Class Size (ECS) *	2		*Lines of Code (LOC)* *	2
				System Design Stability (SDI)	2

* These terms are duplicate in the table.

Based on the results presented in Table 9, we can suggest that most of the high-level quality attributes are quantified through a limited number of metrics, and specifically the Chidamber and Kemerer (1994) and the Li and Henry (1993) suites, suggesting that the same metric can be used for assessing more than one attribute. This is a rather intuitive result in the sense that these are the most-known metric suites. The only metrics outside these suites that appear in the list are: System Design Stability Index (SDI), External Class Complexity (ECC), and External Class Size (ECS). However, we need to clarify that these metrics are only used in two studies each. The most interesting finding concerning this research question is the identification of quality attributes that have not been related to specific metrics. These high-level quality attributes have been omitted from Table 9. A list of the aforementioned quality attributes is provided and described below:

- ***Functionality***. Although investigated in 12 papers and associated with 45 metrics, no metric has been used in more than one paper.
- ***Usability***. Despite the fact that it is studied in 11 papers and is associated with 49 metrics, no metric has been used in more than one paper.
- ***Changeability.*** Although investigated in 11 studies and associated with 66 metrics, one metric (namely: change size) is used in two papers.
- ***Analyzability***. Despite the fact that it is studied in 7 papers and is associated with 29 metrics, no metric has been used in more than one paper.

4.2.3. Validation of software metrics ($RQ_{2.3}$)

With respect to the level of validation, we have classified the studies into four categories: only theoretical validation, only empirical validation, complete validation (i.e., both theoretical and empirical), and no validation (i.e., neither empirical nor theoretical). The frequency of each category is presented in Table 10. We note that the 137 studies involving metrics can be classified to studies that: (a) *introduce* a metric, with or without validating it (56.2%), and (b) *validate* a metric (43.8%). From the results of Table 10, we can observe that only a small portion of the papers have both empirically and theoretically validated the corresponding metrics (8.0%). The majority of studies (i.e., 73.7%) have validated the proposed metrics by employing an empirical research method. More details on specific research methods can be found in Table 11. Moreover, we highlight that the number of papers that perform no validation is non-negligible, i.e., 16.8%. Finally, by focusing only on studies that introduce metrics, we note that approximately ***30% metric introductions are not accompanied with a metric validation.*** We note that this statement does not imply that 30% of metrics have not been validated, since a large number of metrics that have been introduced without evaluation, are later on validated in other, independent, studies.

Table 10
Type of metrics validation.

Validation type	Number of studies		Number of metrics
Complete validation	11	(8.0%)	96
Only empirical	101	(73.7%)	514
Only theoretical	2	(1.5%)	7
No validation	23	(16.8%)	317

Regarding individual metrics, our results suggest that only (11) cohesion and (5) complexity metrics have been validated both theoretically and empirically at the highest level of evidence—i.e., 5 based on the scale of Alves et al. (2010). By considering all levels of evidence, more than 100 metrics have been validated both empirically and theoretically. Among these metrics, only LCOM1, LCOM2, and LCOM5 have been validated in two different studies, increasing the reliability of their assessment.

Next, since the majority of the studies have employed an empirical validation method we further analyze them, with respect to the level of empirical evidence they provide. In particular, in Table 11 we present a mapping of metrics and the level of empirical validation they have received based on Alves et al. (2010) (see Section 3.5). We note that the results presented in Table 11 do not imply the correctness of applying the specific empirical research method, but only the frequency of their application. Although an evaluation of the quality of each study would unveil their rigor and relevance, and potentially provide further useful information, such an investigation falls outside the scope of this mapping study. Usually, studies' quality assessment is part of a systematic literature review, and thus consists an interesting extension of this study. From the results of Table 11, it becomes clear that some metrics are linked to numerous papers, since they have been validated by many studies. For example, we can observe that 25 studies have

Table 11
Mapping between metrics and level of empirical validation evidence.

Level of evidence	Quality metrics	Freq.	Level of evidence	Quality metrics	Freq.
Level 6 (4.46%)	*Depth of Inheritance Tree (DIT)**	2	*Level 3* (19.64%)	*Lack of Cohesion of Methods-1 (LCOM1)**	4
	Number of Children (NOCC)	2		*Lack of Cohesion of Methods-4 (LCOM4)**	3
	*Response for Class (RFC)**	2		*Lack of Cohesion of Methods-5 (LCOM5)**	3
	*Lack of Cohesion of Methods-1 (LCOM1)**	2		*Tight Class Cohesion (TCC)**	3
	*Weighted Methods per Class (WMC)**	2		*Loose Class Cohesion (LCC)*	3
Level 5 (16.07%)	*Lack of Cohesion of Methods-1 (LCOM1)**	7	*Level 2* (17.86%)	*Halstead n1*	2
	*Lack of Cohesion of Methods-2 (LCOM2)**	6		*Halstead n2*	2
	*Cohesion (Coh)**	6		*Input Complexity (Cin)*	1
	*Tight Class Cohesion (TCC)**	6		*Output Complexity (Cout)*	1
	*Lines of Code (LOC)**	5		*Complexity (C)*	1
Level 4 (41.96%)	*Halstead n1*	18			
	Halstead n2	18			
	*Lines of Code (LOC)**	17			
	*Cyclomatic Complexity (CC-VG)**	16			
	*Lack of Cohesion of Methods-1 (LCOM1)**	12			
	*Depth of Inheritance Tree (DIT)**	12			
	Coupling Between Objects (CBO)	9			
	*Response for Class (RFC)**	9			
	Halstead n	7			
	Halstead v	7			

* These terms are duplicate in the table.

validated the relationship between LCOM1 and various QAs. Such a corpus of linked papers can provide a holistic assessment of the capabilities of each metric. Similarly to before, we note that synthesizing this information is out of the scope of this study, since such aggregations are usual an outcome of systematic literature reviews and not mapping studies.

From the 136 studies that involve metrics, 112 provide some form of empirical validation, classified as follows:

- 17.86% of the studies obtained evidence from ***demonstration or working out toy examples***. These studies have in total validated 167 software metrics.
- 19.64% of the studies obtained evidence from ***expert opinions or observations***, validating in total 117 software metrics.
- 41.96% of the studies obtained evidence from ***studies executed in an academic environment***. These studies have in total validated 371 software metrics.
- 16.07% of the studies obtained evidence from ***causal case studies in an industrial context***, validating in total 99 software metrics.
- 4.46% of the studies obtained evidence from ***methods already approved and adopted in industries***. These studies have in total validated 17 software metrics.

4.2.4. Quality metrics and software development phases ($RQ_{2.4}$)

To answer $RQ_{2.4}$, in Table 12, we present the cross-tabulation of development phases and quality metrics. We note that from Table 12, we have excluded phases (i.e., testing and requirements engineering) that are related to very few metrics. In particular, ***testing*** has been associated with 89 metrics, from which only LCOM1 and RFC appear in two studies. Additionally, the ***requirements*** phase has been connected to 73 metrics, from which only the number of use cases appears in two studies.

In the literature, software metrics can be classified into two categories: *design-level* and *code-level* metrics (Al Dallal and Briand, 2012), based on the earliest phase in which they can be calculated. For example, LCOM1 (Lack of Cohesion-1 (Chidamber and Kemerer, 1994)) that requires knowledge on the method body can be calculated at the implementation level, whereas DIT (Depth of Inheritance Tree, (Chidamber and Kemerer, 1994)), which only requires the existence of inheritance relationships can be calculated at the design phase. In the general case the results of Table 12 are in accordance to this definition. However, some interesting findings can be highlighted:

- ***Cyclomatic Complexity (CC)*** (McCabe, 1976), although calculated at source code level, three studies use it at the architectural level, by aggregating methods' cyclomatic complexity to the component level. This practice is quite common in the architecture community, which lacks specific metrics. The lack of architecture level metrics is discussed in the Software Architecture Metrics (SAM) workshop, which is active the last two years (Nord et al., 2014).
- ***Depth of Inheritance Tree (DIT)*** (Chidamber and Kemerer, 1994), ***Number of Children Classes (NOCC)*** (Chidamber and Kemerer, 1994), and ***Number of Methods (NOM)*** (Li and Henry, 1993), although they can be calculated from the design phase, they are also valid at the source code level. Therefore, the studies that employ them are not violating the aforementioned classification.
- ***Response for a Class (RFC)*** (Chidamber and Kemerer, 1994), ***Tight Class Cohesion (TCC)*** (Bieman and Kang, 1995), and ***Loose Class Cohesion (LCC)*** (Bieman and Kang, 1995), need information from the source code level to be calculated. However, some studies are calculating them at the design phase (i.e., from design artifacts).

The rest metrics on ***architecture phase*** (except cyclomatic complexity), are purely architectural metrics that require only high-level design artifacts. Therefore, they can provide the earliest possible quality assessment.

4.2.5. Quality metrics and tools ($RQ_{2.5}$)

Concerning the use of tools that have been employed for the calculation of metrics, we have been able to identify two main categories: (a) tools that have not been assigned a name by their developers—these tools are usually academic ones, or simple prototypes, and (b) named tools. In this mapping study, we have discovered 19 named tools. However, the majority of the studies have used unnamed tools. The list of metrics that are calculated by unnamed tools is presented in the list below. The list refers to unique metrics and not sets of metrics. For example, the first bullet items suggest that there are 5 studies that provide tools to calculate LCOM1, and 5 (different or maybe overlapping) studies that provide tools for NOCC.

- ***5 studies***: Lack of Cohesion of Methods-1 (LCOM1) and Number of Children (NOCC).

Table. 12
List of most frequently metrics for each development phase.

Develop. phase	Quality metrics	Freq.	Develop. phase	Quality metrics	Freq.
Implementation	*Halstead n1*	*20*	*Design*	*Depth of Inheritance Tree (DIT)**	*5*
	Halstead n2	*20*		*Number of Children (NOCC)**	*4*
	Lack of Cohesion of Methods-1 (LCOM1) *	*20*		*Response for Class (RFC)**	*4*
	Lines of Code (LOC) *	*18*		*Tight Class Cohesion (TCC)**	*4*
	Cyclomatic Complexity (CC-VG) *	*12*		*Weighted Methods per Class (WMC)**	*3*
	*Depth of Inheritance Tree (DIT)**	*10*		*Coupling Between Objects (CBO)* *	*3*
	Weighted Methods per Class (WMC) *	*10*		*Loose Class Cohesion (LCC)*	*3*
	Tight Class Cohesion (TCC) *	*9*	*Architecture*	*Cyclomatic Complexity (CC-VG)**	*3*
	Number of Children (NOCC) *	*9*		*Coupling Factor (CF)*	*2*
	Number of Methods (NOM)	*9*		*Coupling Between Objects (CBO)* *	*2*
	Response for a Class (RFC) *	*9*		*Number of Modules*	*2*
	Lack of Cohesion of Methods-5 (LCOM5)	*8*		*Information Flow (Fan-Out)**	*2*
	Message Passing Coupling (MPC)	*8*		*Non-Functional Coverage (NC)*	*2*
Maintenance	*Depth of Inheritance Tree (DIT)**	*2*		*Absolute Adaptability of a Service (AAS)*	*2*
	Weighted Methods per Class (WMC) *	*2*		*Relative Adaptability of a Service (RAS)*	*2*
	Lines of Code (LOC) *	*2*		*Mean of Absolute Adaptability of a Service (MAAS)*	*2*
	Information Flow (Fan-In)	*2*		*Mean of Relative Adaptability of a Service (MRAS)*	*2*
	*Information Flow (Fan-Out)**	*2*		*Level of System Adaptability (LSA)*	*2*

* These terms are duplicate in the table.

Table. 13
List of tools.

Tool Name	Num. of Metrics
Columbus	64
aToucan	61
Tooms	52
Eclipse-based	25
Access tool of the discover environment	22
Fortranal	21
Metrics plug-in	17
The hyss tool	11
The patricia/testmetrics	10
gen++	10
Connecta	6
Open source	6
Solar	5
Together	5
Sourcemonitor	4
Concernmorph	4
irc2m	4
xml-based parser	3
Rest	2

- ***4 studies***: Tight Class Cohesion (TCC), Depth of Inheritance Tree (DIT), Coupling between Objects (CBO), Response for Class (RFC), and Loose Class Cohesion (LCC).
- ***3 studies***: Weighted Methods per Class (WMC), Number of Attributes (NA), Message Passing Coupling (MPC), Number of Methods (NOM), and Cohesion Among Methods in a Class (CAM).
- ***2 studies***: Lines of Code (LOC), Lack of Cohesion of Methods-2 (LCOM2), Lack of Cohesion of Methods-3 (LCOM3), Method-Method through Attributes Cohesion (MMAC), Number of Ancestors (NOA), Data Abstraction Coupling (DAC), and Other Class Method Export Coupling (OCMEC).

The rest of the tools (i.e., those that have been given a name) are summarized in Table 13. Explicitly naming the metrics that each tool is calculating is out of the scope of this mapping study, however, the interested reader can access them in the accompanying technical report (a link is provided in footnote 3 in page 7). From the results of Table 13, we can observe that the majority of authors do not explicitly name the tools that they develop. Although this does not threaten the validity of the original manuscript, it hinders reusability and establishment of metric calculating tools.

5. Discussion

In this section, we discuss the main outcomes of this mapping study by *interpreting our findings*, by *comparing them to related work* (when applicable), *by synthesizing the results of each sub-research question to answer the main RQs*, and by *providing implications to researchers and practitioners*.

5.1. Interpretation of the results

In this section, we provide an interpretation of the obtained results and a comparison to related work. In Section 5.1.1, we discuss the main findings, with respect to quality attributes and metrics, whereas in Section 5.1.2, we discuss the outcomes related to application domains, development phases, programming paradigms, and tools.

5.1.1. Quality attributes and metrics

Based on the findings of our mapping study, we suggest that the most frequently studied quality attributes are the *low-level quality attributes* (i.e., complexity, cohesion, and coupling). This is an expected outcome, because they are considered as cornerstones of the object-oriented programming paradigm, since many patterns and principles (e.g., Gamma et al., 1994; Martin, 2003; Larman, 2004) are based on such QAs. The fact that complexity is more frequently studied than coupling and cohesion is probably due to the popularity of complexity within and outside the object-oriented paradigm, while coupling and cohesion are more popular in object-orientation (they can be defined at module level as well). Finally, a possible explanation on the popularity of cohesion over coupling is the fact that some cohesion metrics (e.g., LCOM) have received criticism and the search of new measures was more intense. On the other hand, the top-studied HL_QA is ***maintainability***. This result can be explained by the already widely accepted importance of maintenance in the software lifecycle. For example, according to van Vliet maintenance costs consume 50 – 75% of the total time / effort budget of a typical software project (van Vliet, 1993). The importance of maintainability can be further emphasized by the existence in the list of some very similar HL_QAs (e.g., changeability, modifiability, etc.) and its sub-characteristics (e.g., stability, analyzability, modularity, adaptability, etc.). The aforementioned findings are supported by related work. In particular, Tahir and MacDonell (2012) also highlighted the frequent occurrence of cohesion, coupling, complexity and maintainability. The importance of main-

tainability is also emphasized by Jabangwe et al. (2004), who focused their SLR on maintainability and reliability.

By further focusing on the quantification of HL_QAs, our results suggested that despite the fact that approximately 2 out of 3 studies link metrics to HL_QA's, only 1 out of 3 combines these metrics in a single calculation formula. This outcome is expected since the accurate assessment of a QA through a single function is a non-trivial task. In the majority of cases, researchers have suggested the use of a weighted sum method (use of regression models) as an aggregation function for combining metrics (Kitchenham, 2010). In the literature, as already discussed by Riaz et al. (2009), multiple models that assess software maintainability through a single function (i.e., 15 studies) have been introduced. Additionally, we note that our results concerning maintainability are in accordance with those of Jabangwe et al. (2004), since both studies suggest that approximately 30% of primary studies use models to measure maintainability. We need to note that metric quantification through a function sometimes receives criticism from research communities that are in favor of investigating association/correlation of metrics to QA.

In addition, a more fine-grained analysis of the used metrics suggests that metrics, which quantify cohesion (e.g., LCOM1, TCC, etc.), size (e.g., LOC, NOM), complexity (e.g., CC, Halstead n1, etc.), inheritance (e.g., DIT and NOCC), and coupling (e.g., CBO, RFC, etc.) are the most commonly employed for quantifying HL_QAs. By contrasting the results on the frequency of metrics to the frequency of quality attributes, we can observe that the most popular metrics are quantifying the most popular QAs. These results are in accordance to Jabangwe et al. (2004), who also underline the importance of these properties. Another interesting observation is that the majority of the aforementioned metrics belong to two well-known metric suites, i.e., Li and Henry (1993) and Chidamber and Kemerer metrics (1994). This observation is also in accordance to related work, since Riaz et al. (2009) suggest that combining metrics of these two suites leads to optimal maintainability prediction models.

Finally, concerning the assessment of the relationship between metrics and quality attributes, the results of this study suggest that in 73% of the cases, an empirical validation is performed. Additionally, only 8% of the papers have performed both an empirical and theoretical validation, whereas only 2 papers have only a theoretical approach. These results are in accordance to Kitchenham (2010), who also acknowledge the higher frequency of empirical validation methods, however with a lower percentage (67%). Furthermore, the results of Kitchenham (2010) suggest that 25% of the papers perform both an empirical and theoretical evaluation. By taking into account that the study of Kitchenham (2010) was performed on papers published before 2005, it is implied that during the last decade empirical validation has gained ground against theoretical validation, since more researchers are focusing on empirical validation, and skip the theoretical part. This phenomenon is more general in the software engineering community, since the awareness of software engineers with respect to empiricism has increased during the last years (Sjøberg et al., 2007). However, we note that the higher numbers of empirical validation can be due to the fact that replication of theoretical validation is not necessary (i.e., if a metric is validated once there is no need to theoretically validate it again). On the other hand, replication of an empirical study is very important in the empirical software engineering community.

5.1.2. Application domains, development Phases, programming paradigms, and tools

In accordance to the previously discussed findings, maintainability is the most frequently studied QA regardless of the application domain or software development phase. The only exceptions to this are the early development phases (i.e., project management and requirements engineering) and some specific application domains (e.g., enterprise and business applications, etc.). Concerning the former, this finding is intuitive in the sense that during these phases software development teams pay more attention in the business aspects of the software (e.g., functionality). Regarding the latter, in domains such as information systems and web applications, the success of the software is related to the amount of functionality that they provide to the user and its low learning curve. Additionally, consistency and analysability are more important for more critical application domains, such as embedded systems and business applications.

We observe that the majority of metrics is calculated at the source code level regardless of the targeted development phase (see Section 4.2.4). This result is in accordance to Riaz et al. (2009), who were able to identify only one study that assesses maintainability using design-level metrics. The preference of software engineering researchers in source code metrics can be attributed to two assumptions: (a) researchers are using detailed-design artifacts that are completely synchronized with the source code, or (b) they calculate source code metrics in earlier development phases to get an approximation of their values as early as possible, by trading-off accuracy with early quality assessment. Concerning programming paradigms, the obtained results have indicated that the majority of metrics are linked to the object-oriented or the functional programming paradigm. This is an expected result in the sense that object-oriented and functional languages are dominating the software engineering domain.[7] In addition, in the literature, we have identified some studies that introduce programming paradigm specific metrics (e.g., as summarized by Abdellatief et al. (2013)), but their number is rather limited.

Finally, concerning metric calculation tools that have been developed in an academic environment, we have observed that 25% of published papers use tools for introducing or evaluating metrics. By comparing the list of metrics that are most frequently used and the lists of metrics that are calculated by tools (as introduced in our primary studies dataset), we observe that object-oriented metrics (e.g., LCOM1, NOCC, CBO, etc.) are calculated by at least 3 identified tools, whereas older metrics, e.g. Halstead n1, n2, etc., have been associated with only one tool (i.e., Fortranal). In particular, regarding LCOM1, five studies have developed their own tools for its calculation, one study used open-source software, and six studies used six different tools developed by others. Therefore, LCOM1 can be calculated by 12 distinct tools. Nevertheless, we need to underline that the list of tools mentioned in this manuscript is not exhaustive, since it only includes tools that have been used for introducing or validating metrics in the literature. Therefore, well-known tools, e.g., Borland Together,[8] SonarQube,[9] etc., are missing from this list, since despite their popularity, they have not been used for research purposes. This would require a separate survey dedicated to identifying tools.

5.2. Synthesis and applicability of the results

In this section, we present a synthesized view of the obtained results through an illustrative example. In particular, we provide an overview of the steps that would be followed by a researcher/practitioner to select the most fitting metrics for assessing the quality of software in the ***maintenance phase*** of an application that is classified as a ***generic*** one (i.e., does not belong to a specific application domain). We note that the purpose of this exam-

[7] This can be evident by the yearly report on the usage of programming languages, as published by TIOBE. See http://www.tiobe.com/tiobe_index.

[8] http://www.borland.com/en-GB/Products/Requirements-Management/Together.

[9] http://www.sonarqube.org/.

ple is not necessarily to directly provide suggestions to researchers / practitioners in metrics selection (since it is a very specific one), but to act as guidance on how to apply the quality assurance / metrics selection process described in Section 1.

In the aforementioned example, the researcher/practitioner would first need to select the quality attributes that he/she wants to focus on (i.e., answering RQ_1 - Which quality attributes should be considered in software development, based on the application domain of the project and the current development phase?). To answer this question, a synthesized view of the results of Table 5 (see Section 4.1.1) and Table 6 (see Section 4.1.2) is required. By cross tabulating the results of the two tables we obtain a matrix, as presented in Fig. 4. To avoid the complexity of such a representation, we mapped two of the dimensions (i.e., application domain and quality attribute) in the horizontal axis of the matrix (i.e., rows), using a nesting. Therefore, using the first two columns we present the most important QAs for each application domain. Then, using the remaining columns, we highlight which QA is important in each development phase. Thus, for the case of the aforementioned example, the red rectangle denotes that the quality attributes that are the most interesting ones in the maintenance phase of applications are ***maintainability***, ***stability***, and ***changeability***. The rest of this example focuses on these QAs.

In Fig. 5, we simulate the process through which the interested researcher/practitioner can select metrics for maintainability, stability, and changeability. The matrix has been obtained by cross tabulating the results of all tables presented in Section 4.2 that aim to answer RQ_2 (i.e., How can we effectively use quality metrics for assessing/quantifying a specific quality attribute?). The rows of the table represent a subset of candidate metrics that can potentially be selected. The presented metrics are a subset of all candidate ones, since in the matrix we have considered only the most frequent ones for each quality attribute. The columns are organized in groups, based on QAs (i.e., we have three groups of columns, one for each QA: maintainability, stability, and changeability). For every group, we have four sub-columns:

- ***Association*** denotes if the specific metric is linked to the QA.
- ***Dev. Phase*** denotes if the specific metric can be calculated from the artifacts of the corresponding development phase.
- ***Level of Evidence*** highlights if the relationship between the specific metric and QA has been empirically validated, and at what level of evidence.
- ***Tool*** suggests if the specific metric can be automatically calculated by a tool.
- Based on the above and by assuming that the quality assessment team is interested in selecting as few metrics as possible, the selection of three metrics is implied by the results of Fig. 5. In particular, ***Lines of Code (LoC)*** and ***Weighted Methods per Class (WMC)*** can be used for assessing maintainability and stability with level of evidence 4 and 3, respectively. Concerning changeability, the team is prompted to use ***Change Size (CS)***, with a level of evidence 3. At this point, we need to note that if a quality assessment team is willing to trade-off the low number of selected metrics with a higher level of evidence, the selection would be different. In particular, more metrics would be added to the matrix, based on Table 9 (see Section 4.2.2).

5.3. Implications for researchers and practitioners

In this section, we discuss the main implications of this study for researchers and practitioners. On the one hand, *concerning researchers*, we have identified some interesting future work directions, and some recommendations for the metric introduction/definition process, as follows:

- ***Quality attributes in need of quantification methods***. The results of this study have unveiled some quality attributes that lack quantification (e.g., change proneness, modularity, maintainability, and stability). Therefore, we suggest researchers to not only associate metrics with these quality attributes, but also develop predictive models that can be used for their quantification.
- ***Quality-driven research in some application domains***. Based on the findings presented in Table 5, we have identified that some application domains are more focused on specific quality attributes than others (e.g., business applications on functionality, embedded systems on correctness, etc.). Thus, we advise researchers interested in these application domains, to focus their research efforts on proposing approaches that safeguard these quality attributes.
- ***Quality assurance in different development phases***. Concerning development phases, the results are two-fold: (a) research on some development phases (e.g., requirements engineering and architecture) is not focused on specific quality attributes; and (b) research on some development phases (e.g., architecture) are reusing metrics calculated from artifacts developed in other phases (e.g., source code) and thus, lack phase-specific metrics. To this end, we highlight the need of introducing metrics that are calculated at the architecting phase, and that quantify requirements-related QAs, e.g. traceability.
- ***Validation of introduced metrics***. Despite the fact that during the last years the empirical validation of metrics has increased, current state-of-research lacks metrics that are validated in both an empirical and a theoretical manner. Therefore, we advise researchers to both mathematically (for those that lack theoretical validation in the past) and empirically validate metrics. Thus, researchers should first assure that the proposed metrics hold some basic metrics properties (e.g., those proposed by Briand et al. (1999)), and then investigate their fitness for the quantification of the targeted QA, following well-known methods (e.g., IEEE Standard for a Software Quality Metrics Methodology (1998)).
- ***Investigation of the quality evaluation/assessment of metrics***. An interesting future research topic that has emerged by analyzing the corpus of primary studies is the thorough investigation of the quality of studies that explore the relationship between metrics and quality attributes. This study should be performed as an SLR (not as a mapping study—as this work is designed) and could potentially explore: (a) all types of validation per metric, (b) whether validation exercises are properly carried out, and (c) linking of papers (introduction and validation).

On the other hand, *concerning practitioners*, we aim at aiding the quality assurance process (an example is provided in Section 5.2). First, based on the application domain and the development phase that they are interested, they can identify quality attributes that are more frequently studied. Next, based on the selected quality attributes, they can exploit the results of Table 9 to proceed in their metric selection process. In addition to that, they can narrow down the vast list of metrics by focusing on metrics that are validated at the highest possible level (see Table 11). Finally, to automate the process of metrics collection, they can consult the accompanying technical report,[3] to explore the list of tools that can be used for their quantification.

6. Threats to validity

In this section, we present the threats to validity that concern our mapping study.

Application Domain	QAs	Development Phases						
		Project Management	Requirements	Architecture	Design	Implementation	Testing	Maintenance
Generic	Maintainability	X	X	X	X	X		X
	Stability	X	X	X	X	X		X
	Testability	X		X	X	X	X	
	Understandability	X	X	X	X			
	Changeability	X	X	X	X	X		X
	Change proneness				X	X		
	Modifiability		X		X	X		
	Functionality	X	X	X		X		
	Usability	X	X	X	X	X		
	Modularity			X		X		
	Reusability	X		X	X	X		
	Analyzability	X		X	X			
	Efficiency	X		X	X			
	Adaptability	X		X				
	Completeness		X					
Web Applications	Functionality	X	X	X		X		
	Maintainability	X	X	X	X	X		X
	Reusability	X		X	X	X		
Embedded Systems	Correctness		X					
	Maintainability	X	X	X	X	X		X
	Traceability	X	X					
	Completeness		X					
	Consistency		X		X			
	Volatility		X					
	Understandability	X	X	X	X			
	Reusability	X		X	X	X		
	Testability	X		X	X	X	X	
	Documentation		X			X		
	Memory Requirements		X					
	Reliability					X		
	Suitability	X		X	X			
Information Systems	Functionality	X	X	X		X		
	Efficiency	X		X	X			
	Recoverability	X		X				
	Maintainability	X	X	X	X	X		X
	Traceability	X	X					
	Understandability	X	X	X	X			
Distributed Systems	Maintainability	X	X	X	X	X		X
	Comprehensibility				X	X		
	Locality			X	X			
	Modifiability		X		X	X		
Database Application	Maintainability	X	X	X	X	X		X
	Analyzability	X		X	X			
	Understandability	X	X	X	X			
	Usability	X	X	X	X	X		
	Accuracy	X		X	X			
	Suitability	X		X	X			
	Soundness				X			
	Consistency		X		X			
	Conciseness				X			
	Cohesiveness				X			
	Validity				X			

Fig. 4. QA selection based on development phase and application domain.

First, we discuss threats to validity that can lead to errors in the primary study identification process (e.g., the selection of digital libraries and search string construction, and study selection bias). Retrieving primary studies from selected publication venues and not by searching in digital libraries, might lead to missing studies that are published in other venues. However, as stated before, the aim of the study is to collect high-quality studies only. One way to achieve this goal is to limit the search to top venues (Kitchenham et al. 2009b) and (Kitchenham et al., 2010). In addition to that, limiting the searching space of the mapping study has not lead to a significant decrease of primary studies, because through the search process we have been able to identify more than 2800 candidate primary studies. In addition to that, the small number of keywords used as a search string, may also lead to missing high-quality stud-

QAs / QMs	Maintainability				Stability				Changeability			
	Association	Dev. Phase	Level of Evidence	Tool	Association	Dev. Phase	Level of Evidence	Tool	Association	Dev. Phase	Level of Evidence	Tool
Depth of Inheritance Tree (DIT)	YES	YES	4	YES								
Lines of Code (LOC)	YES	YES	4	YES	YES	YES	3	YES				
Weighted Methods per Class (WMC)	YES	YES	4	YES	YES	YES	3	YES				
Cyclomatic Complexity (CC-VG)	YES	NO		YES								
Lack of Cohesion of Methods-1 (LCOM1)	YES	NO		YES								
Tight Class Cohesion (TCC)	YES	YES	4	YES								
Number of Children (NOCC)	YES	NO		YES								
Response for Class (RFC)	YES	NO		YES								
Message Passing Coupling (MPC)	YES	NO		YES								
Data Abstraction Coupling (DAC)	YES	NO		YES								
Number of Methods(NOM)	YES	NO		YES								
System Design Stability					YES	NO		NO				
Change Size									YES	YES	3	NO

Fig. 5. Metric selection for the important quality attributes.

ies whose authors have not used common terms for quality and quality-related concepts (quality attributes, quality metrics), or did not use keywords that we would have assumed in the full text of papers. However, we believe that it is highly unlikely for authors not having used the corresponding terms, i.e. “quality attribute” OR “quality characteristic” OR “quality metric” OR “software metric” OR “software measurement” OR “quality requirement” OR “quality framework” OR “non-functional requirement” OR “non-functional requirement”, in the full text of papers that introduce and evaluate quality attributes and metrics.

Second, two authors performed data collection and analysis. One checking the results of the other reduces the possibility of data collection inaccuracies. Also, concerning data synthesis, frequency analysis and cross-tabulation are objective methods less prone to researcher bias. Although, primary studies come from specific venues, we believe that no publication bias exists. The communities that publish in the selected venues cover the whole spectrum of software engineering research.

Finally, concerning repeatability, we believe that this study can be easily replicated by different researchers, since: (a) the mapping study protocol is extensively described in this paper, and (b) there was only limited subjective judgement involved in the data collection and analysis phases.

7. Conclusions

This study aimed at providing a detailed panorama of the state-of-the-art on design time quality attributes and metrics to assess them. To achieve this goal, we performed a systematic mapping study and investigated studies that are published in top software engineering venues. The results of the study suggest that maintainability is the most commonly studied quality attribute, regardless of the application domain or the development phase. High-level quality attributes are most commonly assessed through the Chidamber and Kemerer (1994) and Li and Henry (1993) metrics, without however, an established way of aggregating them in one quality attribute metric score.

The results reported in this study can be useful to both researchers and practitioners. On the one hand, researchers can identify useful research directions (e.g., which quality attributes have not been associated with metrics, yet) and get guidance on how to holistically evaluate metrics that they propose, by performing both an empirical and theoretical validation. On the other hand, practitioners can benefit from this study in their quality assurance planning and their metric selection process. Specifically, we provide a list of the most important quality attributes in specific application domains and development phases, accompanied by the most popular and thoroughly validated metrics for each scenario.

Appendix A. Publication venues

Name	cr.1	cr.2	cr.3	cr.4	Included
IEEE Transactions on Software Engineering	A	Yes	Yes	183	Yes
International Conference on Software Engineering	A	Yes	Yes	118	Yes
IEEE Software	B	Yes	Yes	108	Yes
Software: Practice and Experience	A	Yes	Yes	80	Yes
ACM Transactions on Software Engineering and Methodology	A	Yes	Yes	69	Yes
Journal of Systems and Software	A	Yes	Yes	61	Yes
Information and Software Technology	B	Yes	Yes	46	Yes
European Software *Engineering* Conference and the ACM SIGSOFT International Symposium on the Foundations of Software Engineering	A	Yes	Yes	44	Yes
Automated Software Engineering Conference	A	Yes	Yes	44	Yes
Empirical Software Engineering	A	Yes	Yes	36	Yes

(*continued on next page*)

(*continued*)

Name	cr.1	cr.2	cr.3	cr.4	Included
International Conference on Software Process	A	Yes	Yes	23	Yes
International Symposium on Empirical Software Engineering and Measurement	A	Yes	Yes	21	Yes
ACM Computing Surveys	A	No			No
ACM Transactions on Architecture and Code Optimization	A	Yes	No		No
ACM Transactions on Computer Systems	A	No			No
ACM Transactions on Design Automation of Electronic Systems	A	No			No
ACM Transactions on Embedded Computing Systems	A	No			No
ACM Transactions on Information and System Security	A	Yes	No		No
ACM Transactions on Multimedia Computing Communications and Applications	B	Yes	No		No
ACM Transactions on Programming Languages and Systems	A	Yes	No		No
Acta Informatica	A	Yes	Yes	N/A	No
Computer Standards and Interfaces	B	No			No
Computers and Electrical Engineering	B	No			No
Computers and Security	B	Yes	No		No
Computers in Industry	B	No			No
IBM Journal of Research and Development	A	No			No
IBM Systems Journal	A	No			No
IEEE Transactions on Computers	A	No			No
IEEE Transactions on Dependable and Secure Computing	A	No			No
IEEE Transactions on Multimedia	A	Yes	No		no
IEEE Transactions on Reliability	A	Yes	No		No
IET Computers and Digital Techniques	B	No			No
Industrial Management+Data Systems	B	No			No
Innovations in Teaching and Learning in Information and Computer Sciences	B	No			No
International Journal of Agent Oriented Software Engineering	B	Yes	No		No
International Journal on Software Tools for Technology Transfer	B	Yes	No		No
Journal of Computer Security	B	No			no
Journal of Functional and Logic Programming	B	Yes	No		No
Journal of Object Technology	B	Yes	No		No
Journal of Software	B	Yes	Yes	N/A	No
Journal of Software Maintenance and Evolution: research and practice	B	Yes	No		No
Journal of Systems Architecture	B	Yes	No		No
Journal of Visual Languages and Computing	A	Yes	No		No
Multimedia Systems	B	Yes	No		No
Multimedia Tools and Applications	B	Yes	No		No
Requirements Engineering	B	Yes	No		No
Science of Computer Programming	A	Yes	No		No
Software and System Modelling	B	Yes	No		No
Software Testing, Verification and Reliability	B	Yes	No		No
Text Technology: the journal of computer text processing	B	No			No
Theory and Practice of Logic Programming	A	Yes	No		No
ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication	A	No			No
ACM Conference on Computer and Communications Security	A	No			No
ACM Conference on Object Oriented Programming Systems Languages and Applications	A	Yes	No		No
ACM International Symposium on Computer Architecture	A	Yes	No		No
ACM Multimedia	A	No			No
ACM SIGOPS Symposium on Operating Systems Principles	A	No	No		No
ACM/IFIP/USENIX International Middleware Conference	A	No			No
ACM-SIGACT Symposium on Principles of Programming Languages	A	Yes	No		No
ACM-SIGPLAN Conference on Programming Language Design and Implementation	A	Yes	No		No
Annual Computer Security Applications Conference	A	Yes	No		No
Architectural Support for Programming Languages and Operating Systems	A	Yes	No		No
Aspect-Oriented Software Development	A	Yes	No		No
Conference on the Quality of Software Architectures	A	Yes	No		No
European Conference on Object-Oriented Programming	A	Yes	No		No
European Symposium on Programming	A	Yes	No		No
European Symposium On Research In Computer Security	A	Yes	No		No
Eurosys Conference	A	Yes	No		No
IEEE Computational Systems Bioinformatics Conference	A	No			No
IEEE Computer Security Foundations Symposium	A	Yes	No		No
IEEE International Conference on Software Maintenance	A	Yes	No		No
IEEE International Requirements Engineering Conference	A	Yes	No		No
IEEE/IFIP International Conference on Dependable Systems	A	Yes	No		No
IEEE/IFIP International Symposium on Trusted Computing and Communications	A	No			No
IEEE/IFIP Working Conference on Software Architecture	A	Yes	No		No
IFIP Joint International Conference on Formal Description Techniques and Protocol Specification, Testing, And Verification	A	Yes	No		No
Intelligent Systems in Molecular Biology	A	No			No
International Conference on Compiler Construction	A	Yes	No		No
International Conference on Coordination Models and Languages	A	Yes	No		No
International Conference on Evaluation and Assessment in Software Engineering	A	Yes	Yes	N/A	No
International Conference on Functional Programming	A	Yes	No		No
International Conference on Principles and Practice of Constraint Programming	A	Yes	No		No
International Conference on Reliable Software Technologies	A	Yes	No		No

(*continued on next page*)

(*continued*)

Name	cr.1	cr.2	cr.3	cr.4	Included
International Conference on Security and Privacy for Communication Networks	A	No			No
International Conference on Software Reuse	A	Yes	No		No
International Conference on Virtual Execution Environments	A	No			No
International Symposium Component-Based Software Engineering	A	Yes	No		No
International Symposium on Automated Technology for Verification and Analysis	A	Yes	No		No
International Symposium on Code Generation and Optimization	A	Yes	No		No
International Symposium on High Performance Computer Architecture	A	Yes	No		No
International Symposium on Memory Management	A	Yes	No		No
International Symposium on Software Reliability Engineering	A	Yes	No		No
International Symposium on Software Testing and Analysis	A	Yes	No		No
Tools and Algorithms for Construction and Analysis of Systems	A	Yes	No		No
Usenix Network and Distributed System Security Symposium	A	Yes	No		No
Usenix Security Symposium	A	Yes	No		No
Usenix Symposium on Operating Systems Design and Implementation	A	No			No
USENIX Workshop on Hot Topics in Operating Systems	A	No			No

Appendix B. Primary studies

Abrahao, S. and Poels, G. “A family of experiments to evaluate a functional size measurement procedure for Web applications”, (82:2), 2009, pp. 253 – 269.

Adamov, R. and Richter, L. “A proposal for measuring the structural complexity of programs”, (12:1), 1990, pp. 55 – 70.

Albuquerque, D., Cafeo, B., Garcia, A., Barbosa, S., Abrahão, S. and Ribeiro, A. “Quantifying usability of domain-specific languages: An empirical study on software maintenance”, (101:3), 2015, pp. 245–259.

Al Dallal, J. “Incorporating transitive relations in low-level design-based class cohesion measurement”, (43:6), 2013, pp. 685–704.

Al Dallal, J. “Object-oriented class maintainability prediction using internal quality attributes”, (55:11), 2013, pp. 2028 – 2048.

Al Dallal, J. “Improving the applicability of object-oriented class cohesion metrics”, (53:9), 2011, pp. 914 – 928.

Al Dallal, J. “Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics”, (37:6), 2011, pp. 788 – 804.

Al Dallal, J. and Briand, L. C. “A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes”, (21:2), 2012, Article 8.

Al Dallal, J. and Briand, L. C. “An object-oriented high-level design-based class cohesion metric”, (52:12), 2010, pp. 1346 – 1361.

Alshayeb, M. and Li, W. “An empirical study of system design instability metric and design evolution in an agile software process”, (74:3), 2005, pp. 269 – 274.

Alshayeb, M. and Li, W. “An empirical validation of object-oriented metrics in two different iterative software processes”, (29:11), 2003, pp. 1043 – 1049.

Aranha, E. and Borba, P. “An Estimation Model for Test Execution Effort”, 2007, pp. 107 – 116.

Arisholm, E. “Empirical assessment of the impact of structural properties on the changeability of object-oriented software”, (48:11), 2006, pp. 1046 – 1055.

Arisholm, E. and Sjoberg, D. I. K. “Towards a framework for empirical assessment of changeability decay”, (53:1), 2000, pp. 3 – 14.

Athanasiou D., Nugroho, A., Visser, J. and Zaidman A. “Test Code Quality and Its Relation to Issue Handling Performance”, (40:11), 2014, pp. 1100 – 1125.

Baker, A. and Zweben, S. “A Comparison of Measures of Control Flow Complexity”, (SE-6:6), 1980, pp. 506 – 512.

Baudry, B. and Traon, Y. L. “Measuring design testability of a UML class diagram”, (47:13), 2005, pp. 859 – 879.

Bavota, G., Lucia, A. D., Marcus, A. and Oliveto, R. “Using structural and semantic measures to improve software modularization”, (18:5), 2013, pp. 901 – 932.

Behkamal, B., Kahani, M. and Akbari, M. K. “Customizing ISO 9126 quality model for evaluation of B2B applications”, (51:3), 2009, pp. 599 – 609.

Berenbach, B. and Borotto, G. “Metrics for model driven requirements development”, ACM, New York, NY, USA, 2006, pp. 445 – 451.

Bertoa, M. F., Troya, J. M. and Vallecillo, A. “Measuring the usability of software components”, (79:3), 2006, pp. 427 – 439.

Bieman, J. and Kang, B. K. “Measuring design-level cohesion”, (24:2), 1998, pp. 111 – 124.

Bieman, J. and Ott, L. “Measuring functional cohesion”, (20:8), 1994, pp. 644 – 657.

Black, S. “Deriving an approximation algorithm for automatic computation of ripple effect measures”, (50:7–8), 2008, pp. 723 – 736.

Blaine, J. and Kemmerer, R. A. “Complexity measures for assembly language programs”, (5:3), 1985, pp. 229 – 245.

Bourque, P. and Cote, V. “An experiment in software sizing with structured analysis metrics”, (15:2), 1991, pp. 159 - 172.

Bouwers, E., Deursen, A. v. and Visser, J. “Evaluating usefulness of software metrics: an industrial experience report”, IEEE Press, Piscataway, NJ, USA, 2013, pp. 921–930.

Briand, L., Daly, J. and Wust, J. “A unified framework for coupling measurement in object-oriented systems”, (25:1), 1999, pp. 91 – 121.

Briand, L., Morasca, S. and Basili, V. “Defining and validating measures for object-based high-level design”, (25:5), 1999, pp. 722 – 743.

Briand, L. C., Daly, J. W. and Wüst, J. “A Unified Framework for Cohesion Measurement in Object-Oriented Systems”, (3:1), 1998, pp. 65 – 117.

Breno, M. “A proposal for revisiting coverage testing metrics”, ACM, New York, NY, USA, 2014, pp. 899–902.

Bruntink, M. and van Deursen, A. “An empirical study into class testability”, (79:9), 2006, pp. 1219 – 1232.

Calero, C., Piattini, M. and Genero, M. “Empirical validation of referential integrity metrics”, (43:15), 2001, pp. 949 – 957.

Card, D. and Agresti, W. “Measuring software design complexity”, (8:3), 1988, pp. 185 – 197.

Cazzola, W. and Marchetto, A. “A concern-oriented framework for dynamic measurements”, (57: 1), 2015, pp. 32 – 51.

Chae, H. S., Kwon, Y. R. and Bae, D. H. “A cohesion measure for object-oriented classes”, (30:12), 2000, pp. 1405 – 1431.

Chen, J.-Y. and Lu, J.-F. “A new metric for object-oriented design”, (35:4), 1993, pp. 232 – 240.

Chhabra, J. K., Aggarwal, K. and Singh, Y. “Measurement of object-oriented software spatial complexity”, (46:10), 2004, pp. 689 – 699.

Chidamber, S. and Kemerer, C. “A metrics suite for object oriented design”, (20:6), 1994, pp. 476 – 493.

Cioch, F. A. “Measuring software misinterpretation”, (14:2), 1991, pp. 85 - 95.

Coleman, D., Lowther, B. and Oman, P. “The application of software maintainability models in industrial software systems”, (29:1), 1995, pp. 3 – 16.

Conejero, J. M., Figueiredo, E., Garcia, A., Hernandez, J. and Jurado, E. “On the relationship of concern metrics and requirements maintainability”, (54:2), 2012, pp. 212 – 238.

Coskun, E. and Grabowski, M. “An interdisciplinary model of complexity in embedded intelligent real-time systems”, (43:9), 2001, pp. 527 - 537.

Costello, R. J. and Liu, D.-B. “Metrics for requirements engineering”, (29:1), 1995, pp. 39 – 63.

Dantas, F., Garcia, A. and Whittle, J. “On the role of composition code properties on evolving programs”, ACM, New York, NY, USA, 2012, pp. 291–300.

Davis, J. and LeBlanc, R. “A study of the applicability of complexity measures”, (14:9), 1988, pp. 1366 – 1372.

Dhama, H. “Quantitative models of cohesion and coupling in software”, (29:1), 1995, pp. 65 – 74.

Dromey, R. “A model for software product quality”, (21:2), 1995, pp. 146 - 162.

Durisic, D., Nilsson, M., Staron, M. and Hansson, J. “Measuring the impact of changes to the complexity and coupling properties of automotive software systems”, (86:5), 2013, pp. 1275 – 1293.

Edagawa, T., Akaike, T., Higo, Y., Kusumoto, S., Hanabusa, S. and Shibamoto, T. “Function point measurement from Web application source code based on screen transitions and database accesses”, (84:6), 2011, pp. 976 – 984.

Emam, K. E. and Jung, H.-W. “An empirical evaluation of the ISO/IEC 15504 assessment model”, (59:1), 2001, pp. 23 – 41.

Emam, K. E. and Madhavji, N. H. “An instrument for measuring the success of the requirements engineering process in information systems development”, (1:3), 1996, pp. 201 – 240.

Emerson, T. J. “A discriminant metric for module cohesion”, IEEE Press, Piscataway, NJ, USA, 1984, pp. 294 – 303.

English, M., Buckley, J. and Cahill, T. “Fine-Grained Software Metrics in Practice”, 2007, pp. 295 – 304.

Etzkorn, L., Hughes Jr., W. and Davis, C. “Automated reusability quality analysis of OO legacy software”, (43:5), 2001, pp. 295 – 308.

Etzkorn, L. H., Gholston, S. E., Fortune, J. L., Stein, C. E., Utley, D., Farrington, P. A. and Cox, G. W. “A comparison of cohesion metrics for object-oriented systems”, (46:10), 2004, pp. 677 – 687.

Farbey, B. “Software quality metrics: considerations about requirements and requirement specifications”, (32:1), 1990, pp. 60 – 64.

Feigenspan, J., Apel, S., Liebig, J. and Kastner, C. “Exploring Software Measures to Assess Program Comprehension”, 2011, pp. 127 – 136.

Ferreira, K. A., Bigonha, M. A., Bigonha, R. S., Mendes, L. F. and Almeida, H. C. “Identifying thresholds for object-oriented software metrics”, (85:2), 2012, pp. 244 - 257.

Ferrer, J., Chicano, F. and Alba, E. “Estimating software testing complexity”, (55:12), 2013, pp. 2125 – 2139.

Fernández-Sáez, A. M., Genero, M., Caivano, D. and Chaudron, M. R. V. “Does the level of detail of UML diagrams affect the maintainability of source code? A family of experiments”, (21:1), 2014, pp. 212–259.

Franch, X. and Carvallo, J. “Using quality models in software package selection”, (20:1), 2003, pp. 34 – 41.

Gencel, C. and Demirors, O. “Functional size measurement revisited”, (17:3), 2008, pp. 15:1вЂ“15:36.

Genero, M., Manso, E., Visaggio, A., Canfora, G. and Piattini, M. “Building measure-based prediction models for UML class diagram maintainability”, (12:5), 2007, pp. 517 – 549.

Gold, N., Mohan, A. and Layzell, P. “Spatial complexity metrics: an investigation of utility”, (31:3), 2005, pp. 203 – 212.

Gordon, R. “A Qualitative Justification for a Measure of Program Clarity”, (SE-5:2), 1979, pp. 121 – 128.

Grosser, D., Sahraoui, H. and Valtchev, P. “Predicting software stability using case-based reasoning”, 2002, pp. 295 - 298.

Gui, G. and Scott, P. D. “Ranking reusability of software components using coupling metrics”, (80:9), 2007, pp. 1450 – 1459.

Han, A.-R., Jeon, S.-U., Bae, D.-H. and Hong, J.-E. “Measuring behavioral dependency for improving change-proneness prediction in UML-based design models”, (83:2), 2010, pp. 222 - 234.

Harrison, R., Counsell, S. and Nithi, R. “An evaluation of the MOOD set of object-oriented software metrics”, (24:6), 1998, pp. 491 – 496.

Harrison, R., Counsell, S. and Nithi, R. “Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems”, (52:2–3), 2000, pp. 173 - 179.

Harrison, R., Counsell, S. J. and Nithi, R. V. “An Investigation into the Applicability and Validity of Object-Oriented Design Metrics”, (3:3), 1998, pp. 255 – 273.

Harrison, R., Samaraweera, L., Dobie, M. and Lewis, P. “An evaluation of code metrics for object-oriented programs”, (38:7), 1996, pp. 443 – 450.

Harrison, W. “An entropy-based measure of software complexity”, (18:11), 1992, pp. 1025 - 1029.

He, L. and Carver, J. “Modifiability measurement from a task complexity perspective: A feasibility study”, IEEE Computer Society, Washington, DC, USA, 2009, pp. 430 – 434.

Henry, S. and Kafura, D. “Software Structure Metrics Based on Information Flow”, (SE-7:5), 1981, pp. 510 – 518.

Her, J. S., Kim, J. H., Oh, S. H., Rhew, S. Y. and Kim, S. D. “A framework for evaluating reusability of core asset in product line engineering”, (49:7), 2007, pp. 740 - 760.

Hitz, M. and Montazeri, B. “Chidamber and Kemerer’s metrics suite: a measurement theory perspective”, (22:4), 1996, pp. 267 - 271.

Hordijk, W. and Wieringa, R. “Surveying the factors that influence maintainability: research design”, ACM, New York, NY, USA, 2005, pp. 385 - 388.

Horgan, G. and Khaddaj, S. “Use of an adaptable quality model approach in a production support environment”, (82:4), 2009, pp. 730 - 738.

Huang, S. and Lai, R. “On measuring the complexity of an estelle specification”, (40:2), 1998, pp. 165 - 181.

Jensen, H. and Vairavan, K. “An Experimental Study of Software Metrics for Real-Time Software”, (SE-11:2), 1985, pp. 231 - 234.

Jilani, L., Desharnais, J. and Mili, A. “Defining and applying measures of distance between specifications”, (27:8), 2001, pp. 673 - 703.

Jung, H.-W., Kim, S.-G. and Chung, C.-s. “Measuring software product quality: a survey of ISO/IEC 9126”, (21:5), 2004, pp. 88 - 92.

Jung, H.-W., Pivka, M. and Kim, J.-Y. “An empirical study of complexity metrics in Cobol programs”, (51:2), 2000, pp. 111 - 118.

Kafura, D. and Reddy, G. “The Use of Software Complexity Metrics in Software Maintenance”, (SE-13:3), 1987, pp. 335 - 343.

Kakarontzas, G., Constantinou, E., Ampatzoglou, A. and Stamelos, I. “Layer assessment of object-oriented software: A metric facilitating white-box reuse”, (86:2), 2013, pp. 349 - 366.

Kesh, S. “Evaluating the quality of entity relationship models”, (37:12), 1995, pp. 681 - 689.

Khoshgoftaar, T., Munson, J., Bhattacharya, B. and Richardson, G. “Predictive modelling techniques of software quality from software measures”, (18:11), 1992, pp. 979 - 987.

Koru, A. and Tian, J. “Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products”, (31:8), 2005, pp. 625 - 642. van Koten, C. and Gray, A. “An application of Bayesian network for predicting object-oriented software maintainability”, (48:1), 2006, pp. 59 - 67. van Vliet, H. “Software Engineering: Principles and Practice (3rd Edition)”, Wiley, Chichester, England, 1993.

Kumar Chhabra, J., Aggarwal, K. and Singh, Y. “Code and data spatial complexity: two important software understandability measures”, (45:8), 2003, pp. 539 - 546.

Kusumoto, S., Imagawa, M., Inoue, K., Morimoto, S., Matsusita, K. and Tsuda, M. “Function point measurement from Java programs”, ACM, New York, NY, USA, 2002, pp. 576вЂ“582.

Li, H. F. and Cheung, W. K. “An Empirical Study of Software Metrics”, (SE-13:6), 1987, pp. 697 - 708.

Li, W. “Another metric suite for object-oriented programming”, (44:2), 1998, pp. 155 - 162.

Li, W., Etzkorn, L., Davis, C. and Talburt, J. “An empirical study of object-oriented system evolution”, (42:6), 2000, pp. 373 - 381.

Li, W. and Henry, S. “Object-oriented metrics that predict maintainability”, (23:2), 1993, pp. 111 - 122.

Lindvall, M., Tvedt, R. T. and Costa, P. “An Empirically-Based Process for Software Architecture Evaluation”, (8:1), 2003, pp. 83 - 108.

Loconsole, A. “Empirical Studies on Requirement Management Measures”, IEEE Computer Society, Washington, DC, USA, 2004, pp. 42 - 44.

Lohse, J. B. and Zweben, S. H. “Experimental evaluation of software design principles: An investigation into the effect of module coupling on system modifiability”, (4:4), 1984, pp. 301 - 308.

Losavio, F., Chirinos, L., Matteo, A., Levy, N. and Ramdane-Cherif, A. “ISO quality standards for measuring architectures”, (72:2), 2004, pp. 209 - 223.

Lu, H., Zhou, Y., Xu, B., Leung, H. and Chen, L. “The ability of object-oriented metrics to predict change-proneness: a meta-analysis”, (17:3), 2012, pp. 200 - 242.

Ma, Y., Jin, B. and Feng, Y. “Semantic oriented ontology cohesion metrics for ontology-based systems”, (83:1), 2010, pp. 143 - 152.

Mahmood, S. and Lai, R. “A complexity measure for UML component-based system specification”, (38:2), 2008, pp. 117 - 134.

Masoud, H. and Jalili, S. “A clustering-based model for class responsibility assignment problem in object-oriented analysis”, (93:7), 2014, pp. 110 - 131.

McCabe, T. “A Complexity Measure”, (SE-2:4), 1976, pp. 308 - 320.

Mendes, E., Harrison, R. and Hall, W. “Reusability and maintainability in hypermedia applications for education”, (40:14), 1998, pp. 841 - 849.

Moores, T. T. “Applying complexity measures to rule-based prolog programs”, (44:1), 1998, pp. 45 - 52.

Mouchawrab, S., Briand, L. C. and Labiche, Y. “A measurement framework for object-oriented software testability”, (47:15), 2005, pp. 979 - 997.

Munoz, F., Baudry, B., Delamare, R. and Le Traon, Y. “Usage and testability of AOP: An empirical study of AspectJ”, (55:2), 2013, pp. 252 - 266.

Munson, J. C. and Kohshgoftaar, T. M. “Measurement of data structure complexity”, (20:3), 1993, pp. 217 - 225.

N. Robillard, P. and Boloix, G. “The interconnectivity metrics: A new metric showing how a program is organized”, (10:1), 1989, pp. 29 - 39.

Nesi, P. and Campanai, M. “Metric framework for object-oriented real-time systems specification languages”, (34:1), 1996, pp. 43 - 65.

O’ Cinneide, M., Tratt, L., Harman, M., Counsell, S. and Hemati Moghadam, I. “Experimental assessment of software metrics using automated refactoring”, ACM, New York, NY, USA, 2012, pp. 49 - 58.

Ormandjieva, O., Alagar, V. and Zheng, M. “Early quality monitoring in the development of real-time reactive systems”, (81:10), 2008, pp. 1738 - 1753.

Orme, A., Tao, H. and Etzkorn, L. “Coupling metrics for ontology-based system”, (23:2), 2006, pp. 102 - 108.

Ott, L. M. and Bieman, J. M. “Program slices as an abstraction for cohesion measurement”, (40:11–12), 1998, pp. 691 - 699.

Perepletchikov, M. and Ryan, C. “A Controlled Experiment for Evaluating the Impact of Coupling on the Maintainability of Service-Oriented Software”, (37:4), 2011, pp. 449 - 465.

Perez-Palacin, D., Mirandola, R. and Merseguer, J. “On the relationships between QoS and software adaptability at the architectural level”, (87:1), 2014, pp. 1 - 17.

Pickard, M. M. and Carter, B. D. “A field study of the relationship of information flow and maintainability of COBOL programs”, (37:4), 1995, pp. 195 - 202.

Poshyvanyk, D., Marcus, A., Ferenc, R. and GyimΓ³thy, T. “Using information retrieval based coupling measures for impact analysis”, (14:1), 2009, pp. 5 - 32.

Qu, Y., Guan, X., Zheng, Q., Liu, T., Wang, L., Hou, Y. and Yang, Z. “Exploring community structure of software Call Graph and its applications in class cohesion measurement”, (108:10), 2015, pp. 193–210.

Rama, G., M. and Kak A. “Some structural measures of API usability”, (45:1), 2015, pp. 75 - 110.

Reynolds, R. G. “Metrics to measure the complexity of partial programs”, (4:1), 1984, pp. 75 - 91.

Rising, L. S. and Calliss, F. W. “An information-hiding metric”, (26:3), 1994, pp. 211 - 220.

Rombach, H. “A Controlled Experiment on the Impact of Software Structure on Maintainability”, (SE-13:3), 1987, pp. 344 - 354.

Samson, W., Nevill, D. and Dugard, P. “Predictive software metrics based on a formal specification”, (29:5), 1987, pp. 242 - 248.

Sarkar, S., Maskeri, G. and Ramachandran, S. “Discovery of architectural layers and measurement of layering violations in source code”, (82:11), 2009, pp. 1891 - 1905.

Sarkar, S., Rama, G. and Kak, A. “API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization”, (33:1), 2007, pp. 14 - 32.

Scheller, T. and Kühn, E. “Automated measurement of API usability: The API Concepts Framework”, (61:5), 2015, pp. 145 – 162.

Schwanke, R., Xiao, L. and Cai, Y. “Measuring architecture quality by structure plus history analysis”, IEEE Press, Piscataway, NJ, USA, 2013, pp. 891 - 900.

Sellami, A., Hakim, H., Abran, A. and Ben-Abdallah, H. “A measurement method for sizing the structure of UML sequence diagrams”, (59:3), 2015, pp. 222 – 232.

Serrano, M., Trujillo, J., Calero, C. and Piattini, M. “Metrics for data warehouse conceptual models understandability”, (49:8), 2007, pp. 851 - 870.

Shepperd, M. “Early life-cycle metrics and software quality models”, (32:4), 1990, pp. 311 - 316.

Sjoberg, D. I. K., Anda, B. and Mockus, A. “Questioning software maintenance metrics: a comparative case study”, ACM, New York, NY, USA, 2012, pp. 107 - 110.

Sohn, S. Y. and Mok, M. S. “A strategic analysis for successful open source software utilization based on a structural equation model”, (81:6), 2008, pp. 1014 - 1024.

Sun, X., Leung, H., Li, B. and Li, B. “Change impact analysis and changeability assessment for a change proposal: An empirical study”, (96:10), 2014, pp. 51 – 60.

Thwin, M. M. T. and Quah, T.-S. “Application of neural networks for software quality prediction using object-oriented metrics”, (76:2), 2005, pp. 147 - 156.

Tsaur, W.-J. and Horng, S.-J. “A new generalized software complexity metric for distributed programs”, (40:5–6), 1998, pp. 259 - 269.

Tu, Y. -C., Tempero, E. and Thomborson, C. “An experiment on the impact of transparency on the effectiveness of requirements documents”, accepted for publication, 2015, pp. 1 - 32.

Verelst, J. “The Influence of the Level of Abstraction on the Evolvability of Conceptual Models of Information Systems”, (10:4), 2005, pp. 467 - 494.

Voas, J. M. and Miller, K. W. “Semantic metrics for software testability”, (20:3), 1993, pp. 207 - 216.

Wagner, S., Lochmann, K., Heinemann, L., Kläs, M., Trendowicz, A., Plesch, R., Seidl, A., Goeb, A. and Streit, J. “The quamoco product quality modelling and assessment approach”, IEEE Press, Piscataway, NJ, USA, 2012, pp. 1133–1142.

Wagner, S., Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., Mayr, A., Plösch, R., Seidl, A., Streit, J. and Trendowic, A. “Operationalised product quality models and assessment: The Quamoco approach”, (62:6), 2015, pp. 101 – 123.

Wagner, S., Lochmann, K., Winter, S., Goeb, A. and Klaes, M. “Quality models in practice: A preliminary analysis”, IEEE Computer Society, Washington, DC, USA, 2009, pp. 464–467.

Wang, J., Zhou, Y., Wen, L., Chen, Y., Lu, H. and Xu, B. “DMC: a more precise cohesion measure for classes”, (47:3), 2005, pp. 167 - 180.

Weyuker, E. “Evaluating software complexity measures”, (14:9), 1988, pp. 1357 - 1365.

Woo, G., Chae, H. S., Cui, J. F. and Ji, J.-H. “Revising cohesion measures by considering the impact of write interactions between class members”, (51:2), 2009, pp. 405 - 417.

Yamashita, A. F., Benestad, H. C., Anda, B., Arnstad, P. E., Sjoberg, D. I. K. and Moonen, L. “Using concept mapping for maintainability assessments”, IEEE Computer Society, Washington, DC, USA, 2009, pp. 378–389.

Yau, S. and Collofello, J. “Design Stability Measures for Software Maintenance”, (SE-11:9), 1985, pp. 849 - 856.

Yau, S. and Collofello, J. “Some Stability Measures for Software Maintenance”, (SE-6:6), 1980, pp. 545 - 552.

Yue, T., Briand L. C. and Labiche Y. “aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models”, (24:3), 2015, Article 13.

Zhang, H., Li, Y.-F. and Tan, H. B. K. “Measuring design complexity of semantic web ontologies”, (83:5), 2010, pp. 803 - 814.

Zhang, K. and Gorla, N. “Locality metrics and program physical structures”, (54:2), 2000, pp. 159 - 166.

References

Abdellatief, M., Sultan, A.B.Md., Ghani, A.A.A., Jabar, M.A., 2013. A mapping study to investigate component-based software system metrics. J. Syst. Softw. 86 (3), 587–603.

Abran, A., Moore, J.W., 2004. Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, Los Alamitos, CA.

Alves, V., Niu, N., Alves, C., Valenca, G., 2010. Requirements engineering for software product lines: a systematic literature review. *Inf. Softw. Technol.* Elsevier 52 (8), 806–820.

Al Dallal, J., Briand, L., 2012. A precise method-method interaction-based cohesion metric for object-oriented classes. *Trans. Softw. Eng. Methodol.* Article 8 ACM 21 (2).

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture In Practice. Addison-Wesley, Boston, USA.

Bansiya, J., Davies, C.G., 2002. A hierarchical model for object-oriented design quality assessment. *Trans. Softw. Eng.* IEEE Comput. Soc. 28 (1), 4–17.

Basili, V.R., Caldiera, G., Rombach, H.D., 1994. Goal question metric paradigm. In: Encyclopedia of Software Engineering. John Wiley & Sons, pp. 528–532.

Bieman, J.M., Kang, B., 1995. Cohesion and reuse in an object-oriented system. In: 1st Symposium On Software Reusability (SSR' 95). ACM, pp. 259–262. Seattle, USA, 29 – 30 April.

Briand, L.C., Daly, J.W., Wüst, J.K., 1999. A unified framework for coupling measurement in object-oriented systems. Trans. Softw. Eng. IEEE Comput. Soc. 25 (1), 91–121.

Briand, L.C., Wüst, J.K., 2002. Empirical studies of quality models in object oriented systems. Adv. Comput. 56, 97–166 Elsevier.

Cai, K.Y., Card, D., 2008. An An analysis of research topics in software engineering – 2006. J. Syst. Softw. 81 (6), 1051–1058 Elsevier.

Catal, C., Diri, B., 2009. A systematic review of software fault prediction studies. In: Expert Syst. Appl., 36, pp. 7346–7354. Elsevier.

Chidamber, S., Kemerer, C., 1994. A metrics suite for object oriented design. Trans. Softw. Eng. IEEE Comput. Soc. 20 (6), 476–493.

Eckhardt, J., Vogelsang, A., Fernandez, D.M., 2017. Are “non-functional” requirements really non-functional? An investigation of non-functional requirements in practice. In: International Conference on Software Engineering (ICSE 2016), IEEE Computer Society, pp. 832–842.

Elberzhager, F., Münch, J., Tran, N.N.V., 2012. A systematic mapping study on the combination of static and dynamic quality assurance techniques. Inf. Softw. Technol. 54 (1), 1–15 Elsevier.

Febrero, F., Calero, C., Moraga, M.A., 2014. A systematic mapping study of software reliability modeling. Inf. Softw. Technol. 56 (8), 839–849 Elsevier.

Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P., 2014. Variability in software systems—a systematic literature review. Trans. Softw. Eng. IEEE Comput. Soc. 40 (3), 282–306.

Gamma, E., Helms, R., Johnson, R., Vlissides, J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, USA.

Genero, M., Piattini, M., Calero, C., 2005. A survey of metrics for UML class diagrams. J. Object Technol. 4 (9), 59–92.

Halstead, M.H., 1977. Elements of software science. Elsevier *Science Inc.* USA, New York.

Harrison, R., Counsell, S.J., Nithi, R.V., 1998. An evaluation of the MOOD set of object-oriented software metrics. Trans. Softw. Eng. IEEE Comput. Soc. 24 (6), 491–496.

ISO/IEC 25023:2003, 2003. Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Measurement of System and Software Product Quality. Geneva, Switzerland.

ISO/IEC 9126-1:2001, 2001. Software Engineering – Product Quality (Part 1: Quality model). Geneva, Switzerland.

ISO/IEC/IEEE 24765:2010, 2010. Systems and Software Engineering — Vocabulary. Switzerland, Geneva.

Jabangwe, R., Börstler, J., Šmite, D., Wohlin, C., 2004. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. Empirical Softw. Eng. 20 (3), 640–693 Springer.

Kitchenham, A., 2010. What's up with metrics? A preliminary mapping study. J. Syst. Softw. 83 (1), 37–51 Elsevier.

Kitchenham, B., Brereton, P., Turner, M., Niazi, M., Linkman, S., Pretorius, R., Budgen, D., 2009. The impact of limited search procedures for systematic literature reviews: a participant-observer case study. In: 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, Lake Buena Vista, pp. 336–345. FLorida, 15 – 16 October.

Kitchenham, B., Brereton, P., Turner, M., Niazi, M., Linkman, S., Pretorius, R., Budgen, D., 2010. Refining the systematic literature review process – two participant-observer case studies. Empirical Softw. Eng. 15 (6), 618–653 Springer.

Kitchenham, B., Brereton, P., Budgen, D., Turner, M., Bailey, J., Linkman, S., 2009. Systematic literature reviews in software engineering: a systematic literature review. Inf. Softw. Technol. 51 (1), 7–15 Elsevier.

Kitchenham, B., Pfleeger, S.L., 1996. Software quality: the elusive target. IEEE Softw. IEEE Comput. Soc. 13 (1), 12–21.

Kupiainen, E., Mäntylä, M.V., Itkonen, J., 2015. Using metrics in Agile and Lean Software Development – a systematic literature review of industrial studies. Inf. Softw. Technol. Elsevier 62, pp. 143-163.

Larman, C., 2004. Applying UML and Patterns: An Introduction To Object-Oriented Analysis And Design And Iterative Development, 3rd ed. Prentice Hall, Upper Saddle River, New Jersey, USA.

Li, W., Henry, S., 1993. Object-oriented metrics that predict maintainability. J. Syst. Softw. 23 (2), 111–122 Elsevier.

Martin, R.C., 2003. Agile Software Development: Principles Patterns And Practices. Prentice Hall, Upper Saddle River, New Jersey, USA.

McCabe, T., 1976. A complexity measure. Trans. Softw. Eng. IEEE Comput. Soc. 2 (4), 308–320.

Nord, R.L., Ozkaya, I., Koziolek, H., Avgeriou, P., 2014. Quantifying software architecture quality report on the 1st international workshop on software architecture metrics. *SIGSOFT Softw. Eng. Notes* ACM 39 (5), 32–34.

Oriol, M., Marco, J., Franch, X., 2014. Quality models for web services: a systematic mapping. Inf. Softw. Technol. 56 (10), 1167–1182 Elsevier.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08). Bari, Italy, British Computer Society Swinton, pp. 68–77. 26 – 27 June.

Radjenović, D., Heričko, M., Torkar, R., Živkovič, A., 2013. Software fault prediction metrics: a systematic literature review. In: Inf. Softw. Technol., 55, pp. 1397–1418. Elsevier.

Riaz, M., Mendes, E., Tempero, E., 2009. A systematic review on software maintainability prediction and metrics. In: 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09). IEEE Computer Society, Florida, USA, pp. 367–377. 15-16 October.

Saraiva, J., Barreiros, E., Almeida, A., Lima, F., Alencar, A., Lima, G., Soares, S., Castor, F., 2012. Aspect-oriented software maintenance metrics: A systematic mapping study. In: 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012). IEEE Computer Society, Ciudad Real, Spain, pp. 253–262. 14-15 May.

Sjøberg, D.I.K., Dyba, T., Jorgensen, M., 2007. The future of empirical methods in software engineering research. In: Workshop On the Future of Software Engineering (FOSE '07). IEEE Computer Society, Minneapolis, USA, pp. 358–378. 23 - 25 May.

Tahir, A., MacDonell, S.G., 2012. A systematic mapping study on dynamic metrics and software quality. In: 28th IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society, Riva del Garda, Trento, Italy, pp. 326–335. 23-28 September.

van Koten, C., Gray, A., 2006. An application of Bayesian network for predicting object-oriented software maintainability. Inf. Softw. Technol. 48 (1), 59–67 Elsevier.

van Vliet, H., 1993. Software Engineering: Principles and Practice (3rd Edition). Wiley, Chichester, England.

Vargas, J.A., García-Mundo, L., Genero, M., Piattini, M., 2014. A systematic mapping study on serious game quality. In: 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14), London, UK, pp. 13–14. Article 15, ACMMay.

Zhang, H., Babar, M.A., Tell, P., 2011. Identifying relevant studies in software engineering. Inf. Softw. Technol. 53 (6), 625–637 Elsevier.

Zhou, Y., Xu, B., 2008. Predicting the maintainability of open source software using design metrics. Wuhan Univ. J. Nat. Sci. 13 (1), 14–20 Springer.

Wong, W.E., Tse, T.H., Glass, R.L., Basili, V.R., Chen, T.Y., 2011. An assessment of systems and software engineering scholars and institutions (2003–2007 and 2004–2008). J. Syst. Softw. 84 (1), 162–168 Elsevier.

Elvira Maria Arvanitou is a PhD Student at the University of Groningen, the Netherlands, in the group of Software Engineering and Architecture. She holds an MSc degree in Information Systems from the Aristotle University of Thessaloniki, Greece (2013), and a BSc degree in Information Technology from the Technological Institute of Thessaloniki, Greece (2011). Her research interests include software quality assurance and metrics, software maintainability and stability.

Dr. Apostolos Ampatzoglou is a Guest Researcher in the Johann Bernoulli Institute for Mathematics and Computer Science of the University of Groningen (Netherlands), where he carries out research in the area of software engineering. He holds a BSc on Information Systems (2003), an MSc on Computer Systems (2005) and a PhD in Software Engineering by the Aristotle University of Thessaloniki (2012). His current research interests are focused on reverse engineering, software maintainability, software quality management, open source software engineering and software design. He has published more than 50 articles in international journals and conferences. He is / was involved in over 10 R&D ICT projects, with funding from national and international organizations.

Dr. Alexander Chatzigeorgiou is Professor of Software Engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. He received the Diploma in Electrical Engineering and the PhD degree in Computer Science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. From 1997 to 1999 he was with Intracom S.A., Greece, as a telecommunications software designer. Since 2007, he is also a member of the teaching staff at the Hellenic Open University. His research interests include object-oriented design, software maintenance, and software evolution analysis. He has published more than 100 articles in international journals and conferences. He is a member of the Technical Chamber of Greece.

Dr. Matthias Galster is a Senior Lecturer in the Department of Computer Science and Software Engineering at the University of Canterbury, New Zealand. His current work aims at improving the way we develop high quality software, with a focus on software requirements engineering, software architecture, development processes and practices, and empirical software engineering.

Dr. Paris Avgeriou is Professor of Software Engineering in the Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, the Netherlands where he has led the Software Engineering research group since September 2006. Before joining Groningen, he was a post-doctoral Fellow of the European Research Consortium for Informatics and Mathematics (ERCIM). He has participated in a number of national and European research projects directly related to the European industry of Software-intensive systems. He has co-organized several international conferences and workshops (mainly at the International Conference on Software Engineering - ICSE). He sits on the editorial board of Springer Transactions on Pattern Languages of Programming (TPLOP). He has edited special issues in IEEE Software, Elsevier Journal of Systems and Software and Springer TPLOP. He has published more than 130 peer-reviewed articles in international journals, conference proceedings and books. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution, patterns and link to requirements.