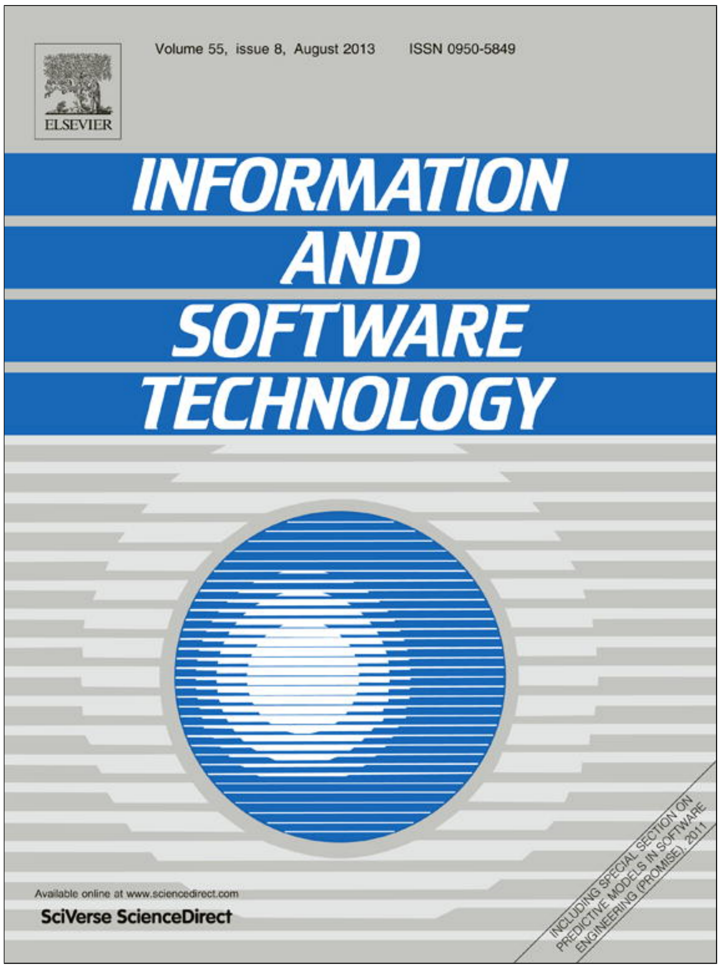
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.


Volume 55, issue 8, August 2013 ISSN 0950-5849
ELSEVIER
INFORMATION AND SOFTWARE TECHNOLOGY
Available online at www.sciencedirect.com
SciVerse ScienceDirect
INCLUDING SPECIAL SECTION ON PREDICTIVE MODELS IN SOFTWARE ENGINEERING (PROMISE), 2011

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/authorsrights

Author's personal copy

Information and Software Technology 55 (2013) 1397–1418

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

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

Software fault prediction metrics: A systematic literature review

Danijel Radjenović [a,b,*], Marjan Heričko [b], Richard Torkar [c,d], Aleš Živkovič [b]

[a] Pro-bit programska oprema d.o.o., Stari trg 15, SI-3210 Slovenske Konjice, Slovenia
[b] University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, SI-2000 Maribor, Slovenia
[c] Blekinge Institute of Technology, SE-371 79 Karlskrona, Sweden
[d] Chalmers University of Technology and University of Gothenburg, SE-412 96 Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 3 September 2012
Received in revised form 13 February 2013
Accepted 14 February 2013
Available online 21 February 2013

Keywords:
Software metric
Software fault prediction
Systematic literature review

ABSTRACT

Context: Software metrics may be used in fault prediction models to improve software quality by predicting fault location.
Objective: This paper aims to identify software metrics and to assess their applicability in software fault prediction. We investigated the influence of context on metrics' selection and performance.
Method: This systematic literature review includes 106 papers published between 1991 and 2011. The selected papers are classified according to metrics and context properties.
Results: Object-oriented metrics (49%) were used nearly twice as often compared to traditional source code metrics (27%) or process metrics (24%). Chidamber and Kemerer's (CK) object-oriented metrics were most frequently used. According to the selected studies there are significant differences between the metrics used in fault prediction performance. Object-oriented and process metrics have been reported to be more successful in finding faults compared to traditional size and complexity metrics. Process metrics seem to be better at predicting post-release faults compared to any static code metrics.
Conclusion: More studies should be performed on large industrial software systems to find metrics more relevant for the industry and to answer the question as to which metrics should be used in a given context.

© 2013 Elsevier B.V. All rights reserved.

Contents

* Corresponding author at: Pro-bit programska oprema d.o.o., Stari trg 15, SI-3210 Slovenske Konjice, Slovenia. Tel.: +386 31 461 290.
E-mail addresses: danijel.radjenovic@pro-bit.si (D. Radjenović), marjan.hericko@uni-mb.si (M. Heričko), richard.torkar@bth.se (R. Torkar), ales.zivkovic@uni-mb.si (A. Živkovič).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.02.009

Author's personal copy

1. Introduction

Fault prediction models are used to improve software quality and to assist software inspection by locating possible faults.[1] Model performance is influenced by a modeling technique [9,16,20,29,30] and metrics [89,138,130,88,66]. The performance difference between modeling techniques appears to be moderate [38,30,60] and the choice of a modeling technique seems to have lesser impact on classification accuracy of a model than the choice of a metrics set [60]. To this end, we decided to investigate the metrics used in software fault prediction and to leave the modeling techniques aside.

In software fault prediction many software metrics have been proposed. The most frequently used ones are those of Abreu and Carapuca (MOOD metrics suite) [1,51], Bansiya and Davis (QMOOD metrics suite) [3], Bieman and Kang [5], Briand et al. [70], Cartwright and Shepperd [74], Chidamber and Kemerer (CK metrics suite) [12,13], Etzkorn et al. [17], Halstead [24], Henderson-Sellers [25], Hitz and Montazeri [26], Lee et al. [37], Li [41], Li and Henry [39,40], Lorenz and Kidd [42], McCabe [44], Tegarden et al. [49]. Many of them have been validated only in a small number of studies. Some of them have been proposed but never used. Contradictory results across studies have often been reported. Even within a single study, different results have been obtained when different environments or methods have been used. Nevertheless, finding the appropriate set of metrics for a fault prediction model is still important, because of significant differences in metrics performance. This, however, can be a difficult task to accomplish when there is a large choice of metrics with no clear distinction regarding their usability.

The aim of this systematic literature review (SLR) is to depict current state-of-the-art software metrics in software fault prediction. We have searched in seven digital libraries, performed snowball sampling and consulted the authors of primary studies to identify 106 primary studies evaluating software metrics. The most commonly used metrics were identified and their fault prediction capabilities were assessed to answer the question of which metrics are appropriate for fault prediction. Ten properties were extracted from the primary studies to assess the context of metrics usage. We identified the most important studies according to study quality and industry relevance. We aggregated the dispersed and contradictive findings to reach new conclusions and to provide directions for practitioners and future research.

The remainder of this paper is organized as follows: Section 2 presents related work. In Section 3 the systematic review method that we used, is described. Section 4 contains the results and answers the research questions. The study is concluded in Section 5.

2. Related work

Kitchenham [34] has published a preliminary mapping study on software metrics. The study was broad and included theoretical and empirical studies which were classified in the following categories: development, evaluation, analysis, framework, tool programs, use and literature survey. The study was later narrowed to 15 studies evaluating metrics against fault proneness, effort and size. Fault proneness was used as a dependent variable in 9

[1] A correct service is delivered when the service implements the system function. A service failure is an event that occurs when the delivered service deviates from the correct/expected service. The deviation is called an error. The adjudged or hypothesized cause of an error is called a fault [2].

Author's personal copy

out of 15 studies. The most frequently used metrics were object-oriented (OO) metrics and, among these, CK metrics.

A systematic review of software fault prediction studies was performed by Catal and Diri [10]. Later, a literature review on the same topic was published [11]. They included all papers (focusing on empirical studies) concerning software fault prediction. They classified studies with respect to metrics, methods and data sets. Metrics were classified in six categories: method-level (60%), class-level (24%), file-level (10%), process-level (4%), component-level (1%) and quantitative-level (1%).

Recently, a review that was similar in design to Catal and Diri's, but more comprehensive in terms of the number of included studies and analyses, was published by Hall et al. [23]. In the review, papers on software fault prediction were included (focusing once again on empirical studies). The main objectives were to assess context, independent variables and modeling techniques. A quantitative model across 19 studies was built to compare metrics performance in terms of *F*-measure, precision and recall. According to the quantitative model, models using OO metrics perform better than those using complexity metrics, while models using LOC perform just as well as those using OO metrics and better than those using complexity metrics. Models using a combined range of metrics performed the best, while models using process metrics performed the worst.

Our study differs from the above reviews in both the aim and scope of the selected studies. The objectives of this review are to assess primary studies that empirically validate software metrics in software fault prediction and to assess metrics used in these studies according to several properties. In the Catal and Diri review, no assessment of software metrics was performed; only metrics distribution was presented. In Hall's review, a quantitative model was presented to assess the raw performance of metrics, without taking into account the context in which the studies were performed. This is most evident in the results of the process metrics, which are reported to be the least successful among all metrics. This might be due to the fact that process metrics are evaluated in post-release software, where faults are rarer and may be more difficult to detect [46,136,115]. Some studies suggest that process metrics are more successful in detecting post-release faults than any static code metrics [88,115,73,85,60].

Of the nine studies that evaluate metrics against faults in Kitchenham's study, only one study was selected by Catal and Diri. As pointed out by Kitchenham, this reflects the different scope of the studies. Catal and Diri were focused on the analysis methods used to construct fault models, whereas Kitchenham's study was focused on using fault models to validate metrics [34]. Our review is different in scope compared to Catal and Diri's, and Hall's study, since we are interested in studies that evaluate metrics and have explicitly excluded studies evaluating modeling techniques. Our review extends Kitchenham's study by continuing from the nine studies evaluating metrics against fault proneness.

Studies evaluating modeling techniques were excluded because they focus on techniques and generally do not contain sufficient information on metrics required by this review. Studies evaluating modeling techniques and evaluating or discussing metrics were included.

3. Research method

In order to summarize the current situation in the field, we have performed a systematic literature review. We followed Kitchenham's guidelines for performing systematic literature reviews in software engineering [32,33] and considered recommendations published in [7,48]. The review design and some of the figures in this section were also inspired by [50].

Following these guidelines, the systematic review was performed in three stages: planning, conducting and reporting the review (Fig. 1). In the first step we identified the need for a systematic review (Step 1, Fig. 1). The objectives for performing the review were discussed in the introduction of this paper. We identified and reviewed existing systematic reviews on the topic in Section 2. None of the previously published reviews were related to our objectives and research questions (Table 1).

The review protocol was developed to direct the execution of the review and reduce the possibility of researcher bias (Step 2) [33]. It defined research questions, search strategy (including search string and digital libraries to search in), the study selection process with inclusion and exclusion criteria, quality assessment, data extraction and the data synthesis process. The review protocol is described in Sections 3.1, 3.2, 3.3, 3.4, 3.5.

The review protocol was developed and evaluated by two researchers (Step 3) and iteratively improved during the conducting and reporting stage of the review.

3.1. Research questions

To keep the review focused, research questions were specified. They were framed with the help of the PICOC criteria [33,45]:

- **P**opulation: software, software application, software system, software project, information system.

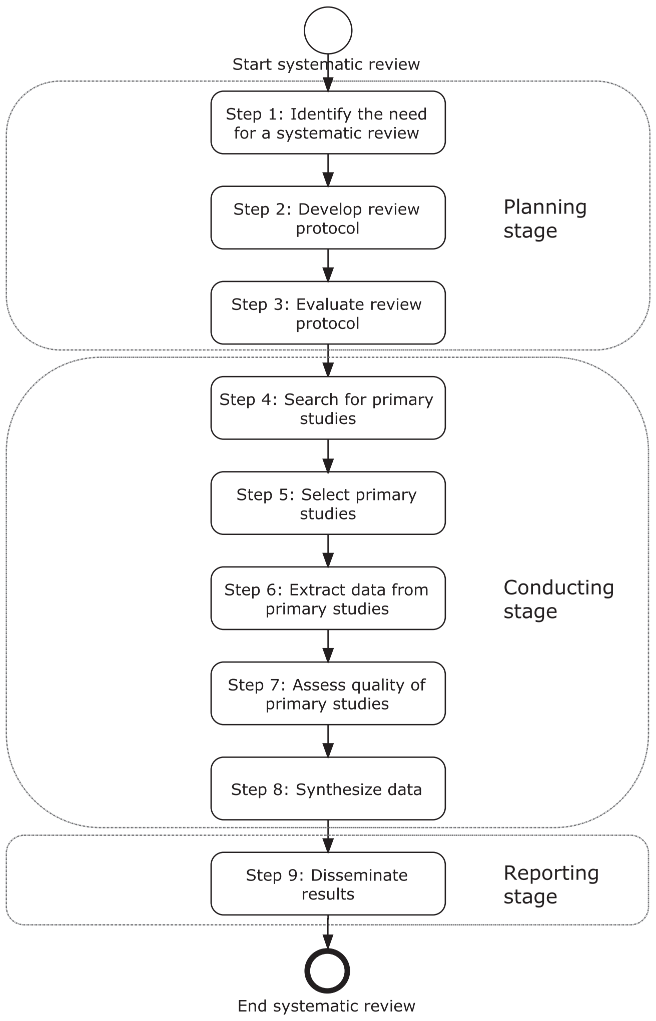

Start systematic review
Step 1: Identify the need for a systematic review
Step 2: Develop review protocol
Step 3: Evaluate review protocol
Planning stage
Step 4: Search for primary studies
Step 5: Select primary studies
Step 6: Extract data from primary studies
Step 7: Assess quality of primary studies
Step 8: Synthesize data
Conducting stage
Step 9: Disseminate results
Reporting stage
End systematic review

Fig. 1. Systematic review steps.

Author's personal copy

Table 1
Research questions.

ID	Research question	Motivation
RQ1	Which software metrics for fault prediction exist in literature?	Identify software metrics commonly used in software fault prediction
RQ2	What kind of empirical validation was performed on the metrics found in RQ1?	Assess the fault prediction performance of software metrics
RQ2.1	Are size metrics useful for fault prediction?	Assess the appropriateness of size metrics as fault predictors
RQ2.2	Are complexity metrics useful for fault prediction? Do they have additional benefits after being adjusted for size?	Assess the appropriateness of complexity metrics as fault predictors and determine whether they correlate with size metrics
RQ2.3	Are OO metrics useful for fault prediction? Do they have additional benefits after being adjusted for size?	Assess the appropriateness of OO metrics as fault predictors and determine whether they correlate with size metrics
RQ2.4	Are process metrics superior to traditional and OO metrics? Is the answer dependent upon the life cycle context?	Assess the appropriateness of process metrics as fault predictors
RQ2.5	Are there metrics reported to be significantly superior in software fault prediction; and if so, which ones?	Identify metrics reported to be appropriate for software fault prediction
RQ2.6	Are there metrics reported to be inferior in software fault prediction; and if so, which ones?	Identify metrics reported to be inappropriate for software fault prediction
RQ3	What data sets are used for evaluating metrics?	Assess the data sets used in the studies and assess the studies' validity
RQ3.1	Are the data sets publicly available?	Determine whether the studies can be repeated and results can be trusted
RQ3.2	What are the size of the data sets?	Investigate the studies' external validity
RQ3.3	What software programming languages are used to implement software from which the data sets are extracted?	Investigate the software metrics domain
RQ4	What are the software development life cycle (SDLC) phases in which the data sets are gathered?	Identify the SDLC phases in which data sets are gathered
RQ4.1	Does the SDLC phase influence the selection of software metrics?	Determine whether different metrics are used in different SDLC phases
RQ4.2	Does software fault prediction accuracy differ across various SDLC phases?	Investigate the effect of an SDLC phase on software fault prediction accuracy
RQ5	What is the context in which the metrics were evaluated?	Assess the context of the studies according to the properties 'Researcher', 'Organization', 'Modeling technique', 'Dependent variable' and 'Dependent variable granularity'

- **I**ntervention: software fault prediction models, methods, techniques, software metrics.
- **C**omparison: n/a.
- **O**utcomes: prediction accuracy of software metrics, successful fault prediction metrics.
- **C**ontext: empirical studies in academia and industry, small and large data sets.

The main goal of this systematic review is to identify software metrics used in software fault prediction (RQ1, Table 1). We searched the literature for publications performing an empirical validation of software metrics in the context of software fault prediction. From the primary studies found, we extracted software metrics to answer RQ1. We analyzed the software metrics to determine which metrics are, and which are not, significant fault predictors (RQ2, R2.1, RQ2.2, RQ2.3, RQ2.4, RQ2.5, RQ2.6). RQ1 and RQ2 were the main research questions, whereas the remaining questions helped us assess the context of the primary studies.

With RQ3 we assessed the data sets used in the primary studies. We were particularly interested in study repeatability (RQ3.1), validity (RQ3.2) and domain, in which the software metrics can be used (RQ3.3).

The software development life cycle (SDLC), also known as the software development process, was first introduced by Benington [4] and was later standardized by ISO/IEC and IEEE [27]. Since there are many SDLC interpretations available, we identified the SDLC phases found in the selected primary studies (RQ4). We then analyzed the influence of the SDLC phases on the selection of software metrics (RQ4.1) and fault prediction accuracy (RQ4.2).

3.2. Search strategy

The search process (Step 4) consisted of selecting digital libraries, defining the search string, executing a pilot search, refining the search string and retrieving an initial list of primary studies from digital libraries matching the search string (Fig. 2).

The databases were selected on the basis of our own experience in conducting a systematic review (third author), recommendations from researchers with experience in SLR and with the help of the universities bibliographers. The largest, most important and relevant databases were selected. Completeness was favored over redundancy, which resulted in a high repetition of studies between databases and a large number of false positives. The initial list of databases was discussed and amended by other researchers with experience in SLR as well as university bibliographers. The selection of databases was not limited by the availability of databases. For the Inspec database, which was not available at the home university, international access was secured.

Fig. 2 presents the selected digital libraries and the number of primary studies found for each database on October 21, 2011. A total of 13,126 primary studies were found in 7 digital libraries. The list was extensive and included many irrelevant studies. However, we decided not to alter the search string to gain less false positives because relevant studies would be missed.

The search string was developed according to the following steps:

1. Identification of search terms from research questions.
2. Identification of search terms in relevant papers' titles, abstracts and keywords.
3. Identification of synonyms and alternative spellings of search terms.
4. Construction of sophisticated search string using identified search terms, Boolean ANDs and ORs.

The following general search string was eventually used: software AND (metric* OR measurement*) AND (fault* OR defect* OR quality OR error-prone) AND (predict* OR prone* OR probability OR assess* OR detect* OR estimat* OR classificat*)

A pilot search was performed. We had a list of nine known relevant studies. When searching digital libraries, eight out of the nine studies were found. One study [67] was not found because the word *metric* was not used in the title or abstract. We tried to adjust the search string, but decided to keep the original one, since the amendment of the search string would dramatically increase the already extensive list of irrelevant studies.

The search string was subsequently adapted to suit the specific requirements of each database. We searched databases by title and

Author's personal copy

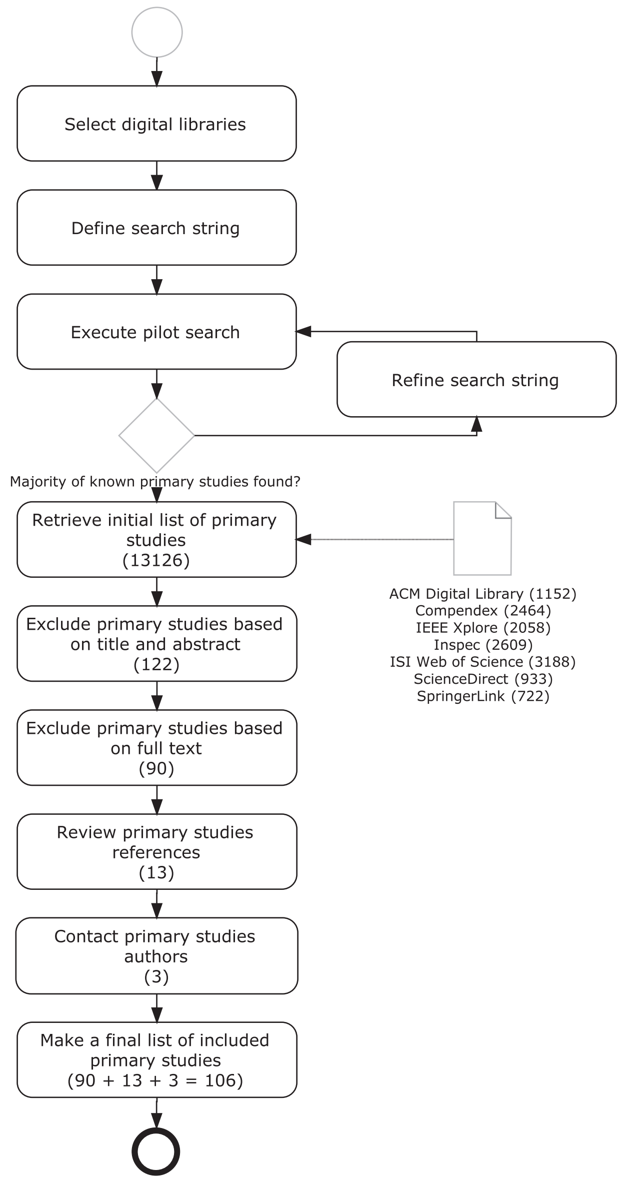

Select digital libraries
Define search string
Execute pilot search
Refine search string
Majority of known primary studies found?
Retrieve initial list of primary studies (13126)
ACM Digital Library (1152)
Compendex (2464)
IEEE Xplore (2058)
Inspec (2609)
ISI Web of Science (3188)
ScienceDirect (933)
SpringerLink (722)
Exclude primary studies based on title and abstract (122)
Exclude primary studies based on full text (90)
Review primary studies references (13)
Contact primary studies authors (3)
Make a final list of included primary studies (90 + 13 + 3 = 106)

Fig. 2. Search and selection of primary studies.

abstract. The search was not limited by the year of publication. Journal papers and conference proceedings were included. The search was limited to English.

3.3. Study selection

During the systematic review, we included empirical studies validating and assessing software metrics in the area of software fault prediction. For the selection of primary studies, the following inclusion and exclusion criteria were used.

Inclusion criteria:

- Empirical studies (academic and industry) using large and small scale data sets `AND`.
- Studies comparing software metrics performance in the area of software fault prediction.

Exclusion criteria:

- Studies without an empirical validation or including experimental results of software metrics in fault prediction `OR`.
- Studies discussing software metrics in a context other than software fault prediction (e.g. maintainability) `OR`.
- Studies discussing modeling techniques (e.g. Naive Bayes, Random Forest) and not software metrics.

Studies discussing and comparing prediction techniques were excluded from the review, since the goal of this review was to assess software metrics and not prediction techniques. Studies evaluating prediction techniques generally do not discuss metrics. They sometimes provide a list of metrics used, but do not evaluate metrics or discuss their role in fault prediction. Studies evaluating prediction techniques and evaluating or discussing metrics were included.

Studies proposing new software metrics and not empirically validating them were also excluded, since we investigated the usefulness of software metrics in predicting software faults; that usefulness is, to one extent, measured by the level of empiricism.

Before performing a study selection, the inclusion and exclusion criteria were tested by two researchers on a random sample of a hundred studies. Although the first results looked promising, there were disagreements among the researchers. A common interpretation was established through dialogue and the inclusion and exclusion criteria were refined.

The study selection process (Step 5) was performed in two steps: the exclusion of primary studies based on the title and abstract and the exclusion of primary studies based on the full text (Fig. 2).

The exclusion of primary studies based on the title and abstract was carried out independently by two researchers. Each developed his own list of selected studies. Table 2 shows the agreement between the two researchers for the first stage. The first researcher included 105, and the second one 89 studies. They agreed in 72 cases to include a study and in 13,004 cases to exclude a study. They did not agree in 50 cases. The final list of selected studies for the first stage was obtained by adding both researchers' selected studies. It included 122 primary studies.

We used inter-rater agreement analysis to determine the degree of agreement between the two researchers. Cohen's Kappa coefficient [14] was calculated. According to the different levels, as stipulated by Landis and Koch [36], the agreement between the two researchers for the exclusion of primary studies based on title and abstract was 'substantial' (0.74).

Full texts were obtained and analyzed for 122 primary studies. In addition to the inclusion and exclusion criteria, the quality of the studies, their relevance to the research questions and study similarity were considered. Similar studies published by the same authors in various journals were removed. 90 primary studies remained after the exclusion of studies based on the full text.

Grey literature was covered by snowball sampling and contacting the authors of primary studies (Fig. 2). The snowball sampling method [21] was used to review the references of the 90 primary studies selected from the databases. It was also used to review the references of the related studies which were not included in the review as primary studies [34,10,11,23]. Additional 13 relevant studies were found and added to the list of primary studies.

Table 2
Researchers' agreement on the exclusion of the primary studies based on title and abstract.

Researcher 2	Researcher 1		Totals
	Included	Excluded	
Included	72	17	89
Excluded	33	13,004	13,037
Totals	105	13,021	13,126

Author's personal copy

We tried to contact all the authors of the primary studies by e-mail to obtain their opinion on whether we missed any studies. Of the 71 e-mails sent, 23 e-mails failed to be delivered, 7 authors replied and 3 new studies were brought to our attention.

Hence, the final list of 106 primary studies was compiled by adding up the primary studies found by searching the databases (90), studies found through snowball sampling (13) and studies found by contacting the authors of primary studies (3). The complete list is provided in the Section 'Systematic review references'. References [72,79,95,98,99,103,106,122,125,126] are used in the analysis but not explicitly referred to in the paper.

3.4. Data extraction

For each of the 106 selected primary studies, the data extraction form was completed (Step 6). The data extraction form was designed to collect data from the primary studies needed to answer the research questions. It included the following properties: 'Title', 'Authors', 'Corresponding email address', 'Year of publication', 'Source of publication', 'Publication type', 'Results', 'Note' and ten properties listed in Table 3 and further discussed in Sections 3.4.1, 3.4.2, 3.4.3, 3.4.4, 3.4.5, 3.4.6, 3.4.7, 3.4.8, 3.4.9, 3.4.10. The properties were identified through the research questions and analysis we wished to introduce. Ten properties were used to answer the research questions (Table 3), while others were used to describe the study (properties 'Title' and 'Authors'); send an email to corresponding authors during the search process (property 'Corresponding email address'); analyze the distribution of the studies over the years (property 'Year of publication'); analyze the number of studies per source (property 'Source of publication'); analyze the number of conference proceedings and journal articles (property 'Publication type'); and analyze the metrics' effectiveness (properties 'Results' and 'Note').

Establishing a valid set of values for all the properties is important in order to summarize the results [33]. A valid set of values was established for the properties 'Data set availability', 'Software development life cycle', 'Researcher', 'Organization', 'Modeling technique', 'Dependent variable' and 'Dependent variable granularity'. The values were chosen on the basis of preliminary research prior to this SLR. For the properties 'Metrics', 'Data set size' and 'Programming language' it was difficult to predict the valid set of values. Therefore, we decided for trivial data extraction, where we extracted data as presented in the studies [48]. After data extraction, a valid set of values based on the gathered data was established for the properties 'Metrics', 'Data set size' and 'Programming language' to enable data synthesis (Step 8).

Table 3
Data extraction properties mapped to research questions and inter-rater agreement.

ID	Property	RQ	Agreement
P1	Metrics	RQ1, RQ2, RQ2.1, RQ2.2, RQ2.3, RQ2.4, RQ2.5, RQ2.6	0.82
P2	Data set availability	RQ3, RQ3.1	0.71
P3	Data set size	RQ3, RQ3.2	0.84
P4	Programming language	RQ3, RQ3.3	1.00
P5	Software development life cycle	RQ4, RQ4.1, RQ4.2	0.80
P6	Researcher	RQ5	0.95
P7	Organization	RQ5	0.62
P8	Modeling technique	RQ5	1.00
P9	Dependent variable	RQ5	0.83
P10	Dependent variable granularity	RQ5	1.00

The data extraction form was piloted on a sample of ten randomly selected primary studies to evaluate sets of values [33]. Because not all studies could fit into predefined values, two new values were introduced. For the property 'Data set availability', the value 'Partially public' was added to account for studies using open source projects, where source code and fault data was publicly available, but not the metrics' values. For the property 'Dependent variable granularity' the value 'Other' was added to support studies with rarely used granularities (e.g. build). After the adjustment, sets of values were suitable for all the 106 selected primary studies and were not altered afterwards.

3.4.1. Metrics (P1)

Our initial goal was to extract all the metrics used in the primary studies. Since there were almost as many metrics used as studies available, we were not able to draw any conclusions from this. Therefore, we decided to categorize the studies according to the metrics used in the following manner:

- **Traditional**: size (e.g. LOC) and complexity metrics (e.g. McCabe [44] and Halstead [24]).
- **Object-oriented**: coupling, cohesion and inheritance source code metrics used at a class-level (e.g. Chidamber and Kemerer [13]).
- **Process**: process, code delta, code churn, history and developer metrics. Metrics are usually extracted from the combination of source code and repository. Usually they require more than one version of a software item.

3.4.2. Data set availability (P2)

The property 'data set availability' was used to assess whether the data sets used were publicly available, which would allow studies to be more easily repeated. If private data sets were used, the study could not be repeated by other researchers in an easy manner, and the results would always be questionable. In our case, we categorized the studies according to data set availability in the following manner:

- **Private**: neither data set, fault data or source code is available. The study may not be repeatable. (If the study did not state the data set availability it was categorized as private in order to make a conservative assessment.)
- **Partially public**: usually the source code of the project and fault data is available, but not the metrics' values, which need to be extracted from the source code and matched with the fault data from the repository (e.g. open source projects like Eclipse and Mozilla). Extracting metrics values and linking them with fault data is not a trivial process. It can lead to different interpretations and errors. Hence, we deem the study as repeatable to some degree only.
- **Public**: the metrics values and the fault data is publicly available for all the modules (e.g. the Promise Data Repository [6]). The study is deemed to be repeatable.

3.4.3. Data set size (P3)

We examined the data set size to determine the external validity of the studies, with the idea being that the larger the data size, the higher the external validity. We are well aware that this is only one aspect of external validity; however, we believe it to be an important aspect. For studies with a small data set size there is a greater possibility that the results have been influenced by the data set and a generalization of the findings may be limited. Therefore, greater validity was given to studies with larger data set sizes.

Data set size was, of course, described differently in the 106 primary studies; mostly by lines of code (LOC), number of classes, files and binaries. We extracted data as specified by the studies and

Author's personal copy

defined data set size categories after the data extraction process. We defined three groups (small, medium and large) and two criterions (LOC and number of classes or files). Categories were defined based on our own experience and the extracted data. If the study specified both LOC and number of classes, the criterion to place the study in a higher group was chosen, e.g. if the study used software with 200 KLOC and 1,100 classes it would be classified as medium according to the KLOC, and as large according to the number of classes. Because we applied a liberal assessment in this case, the study should be classified as large. If the study used more than one data set, the sum (size) of the data sets were considered. When the data set size was not stated, the study was categorized as small per default. The studies were categorized according to the data set size as follows:

- **Small**: less than 50 KLOC (thousands of LOC) OR 200 classes.
- **Medium**: between 50 KLOC OR 200 classes AND 250 KLOC OR 1000 classes.
- **Large**: more than 250 KLOC OR 1000 classes.

3.4.4. Programming language (P4)

We extracted data regarding the programming languages used to implement software from which metrics values were obtained. We assessed which programming languages were most often used in software fault prediction and if the choice of the programming language was associated with a choice of software metrics. Programming languages were extracted as stated and were not categorized, except for rarely used programming languages which were grouped as 'Other'. The 'Not stated' group was introduced for primary studies where the type of programming language was not described.

3.4.5. Software development life cycle (P5)

According to Shatnawi and Li [136], the post-release evolution process of a system is different from the pre-release development process, because the system has been through some rigorous quality assurance checks. Post-release systems tend to have fewer faults than pre-release systems. Faults are also harder to locate and fix in post-release systems.

We decided to use the 'software development life cycle (SDLC)' property to determine whether the SDLC phase influences the selection of software metrics (RQ4.1). We also investigated the effect of the SDLC phase on software fault prediction accuracy (RQ4.2). The studies were categorized according to SDLC into: **pre-release** and **post-release**. Pre-release SDLC included all studies using software in the design, implementation or testing phase. The studies performing validation on software used in production, or those having more than one version of software, were classified as post-release. If SDLC was not stated, the study was categorized as pre-release.

3.4.6. Researcher (P6)

To rank studies we believed to be especially relevant to the industry we used the properties 'Researcher' (P6) and 'Organization' (P7). With these two properties we want to point out research performed by researchers coming from the industry as well as research performed in an industrial setting. We believe that the ultimate goal of every study validating metrics is to find metrics, which can later be used in the industry. Therefore, it is essential that metrics are validated also in the industry.

The 'Researcher' property was used to analyze the share of research performed by researchers coming from the industry. Studies were categorized according to the authors' affiliation with either **academia** or **industry**. When authors of the same paper came from academia and from the industry, the first author was considered.

3.4.7. Organization (P7)

The property 'Organization' was introduced because many studies were performed by researchers coming from academia working in an industry setting. As with the property 'Researcher', the focus here was on studies performing validation on the data sets from the industry. With this property we categorized studies with regard to the origin of the data set into: **academia** and **industry**. When the domain was not stated, the study was categorized as academic, in order to conduct a more conservative assessment.

3.4.8. Modeling technique (P8)

The property 'Modeling technique' was used to examine the influence of the modeling technique on the selection of metrics. The studies were categorized into: **statistical** and **machine learning**. The 'statistical' category included all the statistical models like logistic and linear, univariate and multivariate regression, whereas the 'machine learning' category included machine learning and genetic algorithms. (The modeling techniques are covered in more detail in [23,10,11].)

3.4.9. Dependent variable (P9)

To extract data about dependent variables we introduced the properties 'Dependent variable' (P9) and 'Dependent variable granularity' (P10).

The studies were categorized, based on the dependent variable, into: **detecting**, **ranking** and **severity**. The 'detecting' category included studies classifying modules as fault-prone or not. The 'ranking' category included studies ranking modules according to the number of faults they exhibited while the 'severity' category included studies classifying modules into different fault severity levels.

3.4.10. Dependent variable granularity (P10)

We used the property 'Dependent variable granularity' to determine the level at which predictions were made. Predictions carried out on a lower level of granularity were deemed to be more useful than ones carried out on larger modules, since it enabled more accurate predictions of fault location. Studies were classified according to the 'Dependent variable granularity' into the following categories: **Method**, **Class**, **File**, **Package** and **Other**.

3.5. Study quality assessment

In addition to the inclusion/exclusion criteria, the quality of each primary study was assessed by the quality checklist for quantitative studies. The quality checklist questions were developed by suggestions summarized in [33]. Each question in the quality checklist was answered with 'Yes', 'Partially' or 'No', and marked by 1, 0.5 and 0 respectively. The final score of the study ranged from 0 to 20, where 0 is the lowest score, representing lower quality, and 20 is the highest score, representing high quality studies, according to our definitions. A cutoff value for excluding a study from the review was set at 10 points. Since the lowest score for the study was 11, all the studies were included on the basis of the quality checklist.

All 20 questions, with a summary of how many studies were marked with 'Yes', 'Partially' and 'No' for each question, are presented in Table B.6 (see appendix). Most of the studies were successful at answering questions regarding metrics Q4, Q5 and Q6, with the exception being question Q5, where 9% of the studies did not fully define the metrics they used. The metrics were considered as defined if it was possible to establish the definition or origin of the metrics. Sometimes only the number of metrics or a group of metrics was given, without specifying or referencing the metrics. Most of the studies poorly defining the metrics were, for some reason, published in conference proceedings.

Author's personal copy

Questions regarding the data sets Q2, Q7, Q8 and Q20 were rated lower. The data set size (Q2) was considered not justified if it was not stated or it was too small. 25% of the studies failed to meet this request. Many studies failed to report the data set size. In [150,100] the authors mention a very large telecommunication system, but its size was never specified. In [152,80,140,148] a small data set with less than 100 classes was used, making the studies' validity questionable, since the conclusions were based on a small sample.

Only 53% of data sets were adequately described (Q8) when listing information about data set size, availability, programming language used and context. This information is important when comparing studies and when deciding on using a specific set of metrics in a system, since not every metric is suitable for every system. The question with the lowest rate was data set availability and repeatability of the studies (Q20). Only 21% of all these studies used publicly available data sets, whereas 58% of the studies used a private data set. The importance of data set availability in discussed in more detail in Section 4.4.1.

There was a noticeable difference in rates between journal papers and papers published in conference proceedings. Conference proceedings scored slightly worse, because they failed to report all the details, which could be due to space limitations.

For the interested reader we suggest taking into consideration four studies in particular, [113,137,76,153], which scored the maximum of 20 points.

Readers from the industry, searching for metrics to implement in their systems, may be especially interested in 14 studies [117,120,121,119,118,124,130,131,128,129,147,146,149,92] performed by researchers in the industry and in an industrial setting. They were performed on large data sets, but unfortunately private ones. The exception are two studies [128,129], which were performed using partially available data sets, and [149], which used a small data set.

3.6. Validity threats

3.6.1. Publication bias

Publication bias is the tendency to publish positive results over negative results, where studies presenting positive results are more likely to be published than studies with negative results [33]. Although one of our objectives was to find successful software fault prediction metrics, we did not limit our study prior to the search and selection process. On the contrary, we searched for all studies performing an empirical evaluation of software metrics in software fault prediction, which was our only criterion. In addition, we tried to address this issue by searching conference proceedings and Grey Literature, contacting the authors of the selected studies and by using snowball sampling.

The authors of the primary studies could be biased when selecting metrics. This might be the reason for a higher number of studies validating OO metrics rather than traditional size and complexity metrics. The selected studies are from time period when many organizations were adopting or already using OO systems. The broad acceptance of the OO development paradigm could have influenced the authors to use OO metrics in greater extent.

3.6.2. Searching for primary studies

The search strategy was designed to find as many studies as possible. We constructed a very wide search string, which resulted in 13,126 studies found in seven databases. Although the result list was extensive and included a lot of false positives, we decided to keep the original search string in order not to miss any potentially relevant studies.

Our additional search strategy, snowball sampling, resulted in an additional thirteen relevant studies and, finally, three studies were discovered by contacting the authors themselves. Out of the total of 16 studies, we were somewhat relieved to discover that seven studies could not be found in any of the seven databases. The remaining nine studies could be found in the databases but were not captured by our already extensive search string.

We concluded that searching for primary studies in databases is neither efficient nor effective, which is primarily due to inconsistent terminology [19] (similar observations have been made in [33,48,7]).

In addition to searching databases, we used alternative search strategies to minimize the possibility of missing potentially relevant studies. However we cannot exclude the possibility that we overlooked a relevant study.

3.6.3. Study selection

The search and exclusion of studies based on the title and abstract (13,126 studies in total), was carried out independently by two researchers. First, a pilot selection process was performed on 100 randomly selected studies to establish a common ground of understanding regarding the inclusion/exclusion criteria and to find and resolve any potential disagreements. After the pilot search, the inclusion/exclusion criteria were refined.

Each researcher developed his own list of selected studies. An inter-rater agreement analysis was then used to determine the degree of agreement between the two researchers (Table 2), which was substantial, according to [36].

A final exclusion, based on full text, was carried out by a single researcher. There is therefore a possibility that the exclusion was biased and that a study may have been incorrectly excluded.

3.6.4. Data extraction and quality assessment

Due to time constrains and the large number of studies to review, data extraction and quality assessment were performed by one researcher and checked by a second researcher, as suggested in [7,48]. In our case, the second researcher performed data extraction and quality assessment on ten randomly selected studies.

As in the study selection process we used an inter-rater agreement analysis to calculate the Cohen's Kappa coefficient. Agreement between the two researchers for each property is presented in Table 3. The researchers were in agreement on extracting the programming language (P4), modeling technique (P8) and dependent variable granularity (P10). There were some misunderstandings about extracting data with multiple choices and incomplete data, which were later discussed and agreed upon. Eight out of the ten properties had a Cohen's Kappa coefficient of 0.80 or higher, which is substantial, according to Landis and Koch [36].

The most bias was expected in the quality assessment, since questions may be hard to answer objectively. None of the ten studies assessed by two researchers got the same final score. However, the most significant difference in the score for an individual study between the two researchers was two. The quality checklist may not be the most precise form of quality assessment, but we believe it is solid enough to distinguish between better and weaker studies.

4. Results

In this section the selected primary studies are described and the results are provided. A short overview of the studies' distribution over the years, and per source, is presented in Section 4.1. In Sections 4.2, 4.3, 4.4, 4.5, 4.6 research questions are answered.

Author's personal copy

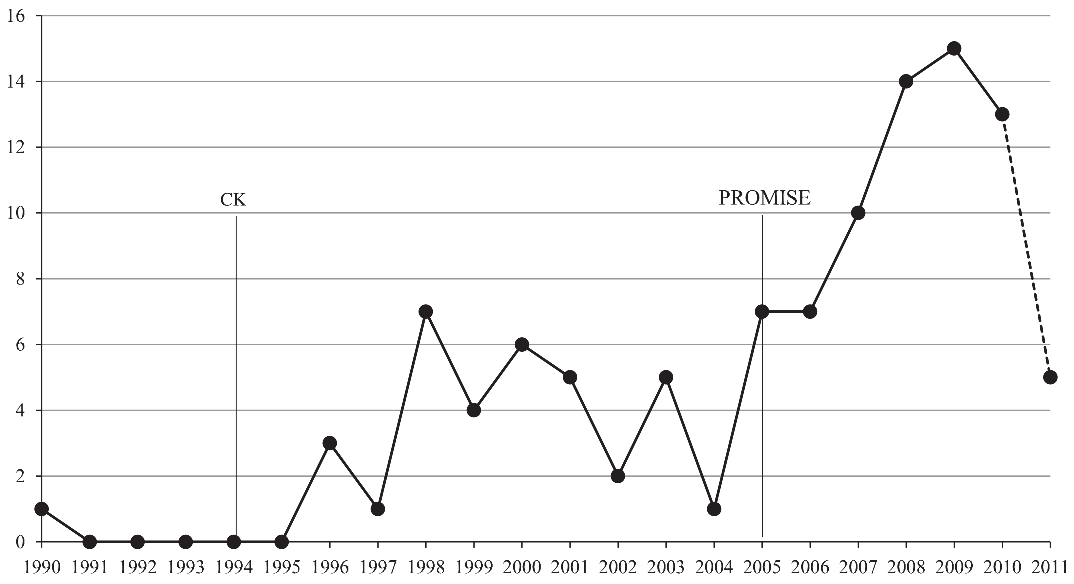


Fig. 3. The distribution of studies over the years.

4.1. Primary studies

In this review, we have included 106 primary studies that evaluate the performance of software metrics fault prediction. In Section 4.1.1 the distribution over the years is presented to demonstrate how the interest in metrics validation has changed over time. The most dominant publication sources are presented in Section 4.1.2.

4.1.1. Publication year

One of the oldest and well-known metrics presented by McCabe [44] dates back to 1976. In 1991, Chidamber and Kemerer [12] presented the CK metrics suite and refined it in 1994 [13]. Although we found a few studies from the 1980s, none were included in the review. In Fig. 3 it can be seen that after the CK metrics suite was introduced, many studies validating software metrics were published. From 1996 until 2005, an average of four studies per year were published. In 2005, when the PROMISE repository [6] was released, the number of studies per year started to increase, reaching 14, 15 and 13 studies per year in the years 2008, 2009 and 2010, respectively. Data is not consistent for the year 2011, since the year was not yet finished when we started searching for studies in October 2011; it takes up to half a year for studies to appear in databases. Nevertheless, five studies from 2011 were included. Since 2005, 71 (67%) studies were published, indicating that we have included more contemporary and relevant studies. It also shows that the software metrics research area is still very much relevant to this day.

4.1.2. Publication source

The number of selected studies per source is shown in Table 4. Only sources having three or more publications are listed. The most dominant journal, with 19 publications, is IEEE Transactions on Software Engineering, followed by the International Conference on Software Engineering and the PROMISE conference. The first three sources contain 38% and the first ten sources contain 66% of all the selected studies. This makes searching for studies about software metrics easier, since there are journals and conferences where studies such as these are published in most cases.

There were slightly more studies presented at conferences (59) than in journals (47).

Table 4
Number of studies per source and the cumulative percent of studies for top sources.

Source	#	Σ %
IEEE Transactions on Software Engineering	19	18
International Conference on Software Engineering	13	30
International Conference on Predictive Models in Software Engineering	8	38
Empirical Software Engineering	5	42
Information and Software Technology	5	47
International Software Metrics Symposium	5	52
Journal Of Systems and Software	5	57
International Symposium on Software Reliability Engineering	4	60
European Conference on Software Maintenance and Reengineering	3	63
Software Quality Journal	3	66

4.2. RQ1: What software metrics for fault prediction exist in literature?

In 61 (49%) studies, the object-oriented metrics were used, followed by traditional metrics, which were used in 34 (27%) studies. Object-oriented metrics were used twice as often when compared with traditional metrics. There were many studies comparing fault prediction performance of OO and traditional metrics. They used and validated OO metrics but had traditional metrics (e.g. LOC) just for comparison. As we have seen from the distribution of studies over the years, all of the studies except one were published after the CK metrics suite was introduced, which has had a significant impact on metrics selection ever since.

Process metrics were used in 30 (24%) studies. It is interesting to note that there is no significant difference in the number of studies using process or traditional metrics.

After 2005, the use of process metrics (30%) has slightly increased and the use of OO metrics (43%) has slightly decreased. The number of studies using traditional metrics (27%) has remained the same.

The most frequently used metrics are the [13,12] (CK). According to our primary studies, the CK metrics were validated for the first time in 1996 [62] and most recently in 2011 [75,82,76,105]. They were also the first validated metrics in our selected studies with the exception of [102]. Their popularity is evenly distributed over the years and there is no sign that this will change in the

Author's personal copy

future. Out of 106 selected studies, CK metrics were used in half of them. The most commonly used metrics from the CK metrics suite include: NOC (53), DIT (52), RFC (51), LCOM (50), CBO (48) and WMC (44). (The number in parenthesis indicates the number of times the metrics were used used.)

The WMC metric ('Weighted Methods per Class') is defined as the sum of methods complexity [12,13]. But, method complexity was deliberately not defined in the original proposal in order to enable the most general application of the metric [13]. To calculate method complexity, any traditional static size or complexity metric can be used. In this review, we did not make any distinction between different implementations of WMC because studies do not always report how the metric was calculated.

There are three common implementations of the WMC metric found in the primary studies:

- Simple WMC: The complexity of each local method is considered to be unity. The WMC is equal to the number of local methods in a class. However, it should not be mistaken for the NLM metric ('Number of Local Methods'), which is the number of local methods defined in a class that are accessible outside the class (e.g. public methods) [41]. The NLM metric was used in [154,57,94,129,136]. Second similar metric is the NOM metric ('Number Of Methods'), which is the count of all methods defined in a class (including inherited methods) [3] and was used in [156,61,58,77,76,153,128,94,105,136]. Another similar metric is the NMC metric ('Number of Methods per Class'), which is defined as the number of public, private and protected methods declared in a class, but does not include inherited methods [15]. It was used in [149,83,87]. When counting methods in a class, the WMC is essentially a size measure.
- WMC_LOC: The complexity of each local method is calculated using the LOC metric. The WMC is equal to the sum of lines of code of all local methods in a class. In this case, WMC is a size measure.
- WMC_McCabe: The complexity of each local method is calculated using McCabe's cyclomatic complexity. The WMC is equal to the sum of McCabe's cyclomatic complexity of all local methods in a class. In this case, the WMC is a complexity measure.

Therefore, the WMC is not really an OO metric, but it is more of a size or complexity metric, depending on implementation.

McCabe's cyclomatic complexity [44] was the most frequently used of the traditional metrics. It was used in 43 studies. Other popular metrics include LCOM1 by Henderson-Sellers [25] (14); NOA by Lorenz and Kidd [42] (13); NOM by Bansiya and Davis [3] (12); LCOM3 by Hitz and Montazeri [26] (11); N_1, N_2, η_1 and η_2 by Halstead [24] (12); ACAIC, ACMIC, DCAEC, DCMEC and OCMEC by Briand et al. [70] (11); TCC and LCC by Bieman and Kang [5] (10).

The extensive list of the most commonly used metrics in the selected studies is presented in appendix, Table C.7.

4.3. RQ2: What empirical validation was performed on the metrics found in RQ1?

This section presents the results of metrics evaluation found in the selected studies. The studies are grouped according to software metrics into: size, complexity, OO and process metrics.

The overall effectiveness of metrics and effectiveness in regard to SDLC, size and programming language is summarized in Table 5. In overall effectiveness, all the studies were taken into account to get a general assessment. To investigate which metrics were effective in a particular environment, we assessed how the metrics' effectiveness was influenced by pre- and post- release SDLC, small and large data sets, procedural and OO programming languages.

Table 5
Metrics' effectiveness in terms of SDLC, size, programming language and overall.

Metric	Overall	SDLC		Size		Prog. language	
		Pre	Post	Small	Large	Proc	OO
LOC	+	++	+	++	+	+	++
N_1	0	+	0	+	0	0	0
N_2	0	+	0	+	0	0	0
η_1	0	0	0	0	0	0	0
η_2	0	0	0	0	0	0	0
CC	+	0	+	0	+	0	+
CBO	++	++	++	++	++		++
DIT	0	+	0	+	0		0
LCOM	0	+	0	+	0		0
NOC	0	–	0	–	0		0
RFC	++	++	++	++	++		++
WMC	++	++	++	++	++		++
Delta	+		+		+	0	+
Churn	++		++		++	++	++
Developer	+		+		+	0	++
Past faults	+		+		+	0	+
Changes	++		++		++	++	++
Age	++		++		++	++	++
Change set	++		++		++	+	++

For the category 'size', medium and large data sets were merged, as there were not many studies using medium-sized data sets. This way, a clear distinction between studies using small and large data sets was made.

Metrics were selected based on their frequency of use in the selected primary studies. Including more metrics in the table was not possible due to a lack of use of other metrics. Nevertheless, we believe the selected metrics represent each of the metrics' categories. The LOC metric was selected as the representative of size metrics, as other size measures were rarely used. Complexity metrics were described by McCabe's cyclomatic complexity CC and Halstead's total number of operators N_1, total number of operands N_2, number of unique operators η_1 and number of unique operands η_2. Chidamber and Kemerer metrics (CBO, DIT, LCOM, NOC, RFC and WMC) were selected as the most frequently used object-oriented metrics, since other OO metrics, like MOOD and QMOOD, were rarely used. Process metrics were represented by code delta, code churn, the number of developers, the number of past faults, the number of changes, the age of a module and the change set size.

Effectiveness was assessed with a five-point scale to depict the degree of the metrics' effectiveness. It was denoted by the symbols ++, +, 0, – and – –, where ++ indicates a strong positive, + a weak positive, 0 no, – a weak negative and – – a strong negative correlation between the metrics and software fault prediction. A blank entry indicates that the estimation of a metric's effectiveness could not be made, because there were not any (or not enough) studies in the category. When assessing the metrics' effectiveness, the reliability of the studies was considered by taking into account the results of a study quality assessment. Greater reliability was given to the studies with a higher quality assessment score. The results of the studies with higher reliability were considered to a greater extent than the results of the studies with lower reliability.

4.3.1. RQ2.1: Are size metrics useful for fault prediction?

The simplest, the easiest to extract and the most frequently used metric, i.e. LOC, is still being discussed to this day. There are many studies investigating the relationship between lines of code and number of faults. The simplest studies have ranked the modules according to their size to find out whether a small number of large modules are responsible for a large proportion of faults. E.g. in Zhang [151] three versions of Eclipse were used to investigate pre-release and post-release ranking ability of LOC at the

Author's personal copy

package level. This study showed that 20% of the largest modules were responsible for 51–63% of the defects.

Ostrand et al. [130] used the negative binomial regression model on two large industrial systems. In the simple model, using only LOC, the percentage of faults, contained in the 20% of the files that were the largest in terms of the number of lines of code, was on average 73% and 74% for the two systems. In a richer model, where other metrics were used, the top-20% of files ordered by fault count contained, on average, 59% of the lines of code and 83% of the faults. The top-20% of files contained many large files, because the model predicted a large number of faults in large files. In analyzing which files were likely to contain the largest number of faults relative to their size, they used the model's predicted number of faults and the size of each file to compute a predicted fault density. The top-20% of files contained, on average, only 25% of the lines of code and 62% of the faults. Sorting files by predicted fault density was not as effective as sorting files according to fault count at finding large numbers of faults, but it does result in considerably less code in the end.

Fenton and Ohlsson [84] investigated, among many hypotheses, the Pareto principle [28] and the relationship between size metrics and the number of faults. They used a graphical technique called the Alberg diagram [127] and two versions of a telecommunication software. As independent variables LOC, McCabe's cyclomatic complexity and SigFF metrics were used. In pre-release 20% of the modules were responsible for nearly 60% of the faults and contained just 30% of the code. A replicated study by Andersson and Runeson [59] found an even larger proportion of faults, in a smaller proportion of the modules. This result is also in agreement with [130,151].

Fenton and Ohlsson also tested the hypothesis of whether size metrics (such as LOC) are good predictors of pre-release and post-release faults in a module and whether they are good predictors of a module's pre-release and post-release fault density. They showed that size metrics (such as LOC) are moderate predictors of the number of pre-release faults in a module, although they do not predict the number of post-release failures in a module, nor can they predict a module's fault density. Even though the hypothesis was rejected, the authors concluded that LOC is quite good at ranking the most fault-prone modules. Andersson and Runeson, on the other hand, got varying results. The first two projects did not indicate any particularly strong ranking ability for LOC. However, in the third project, 20% of the largest modules were responsible for 57% of all the faults.

Koru et al. [35,104] reported that defect proneness increases with size, but at a slower rate. This makes smaller modules *proportionally* more problematic compared with larger ones.

In [34,59], it is noted that relating size metrics to fault density may be misleading, as there will always be a negative correlation between size metrics and fault density, because of the functional relationship between the variables [47]. However, no studies using fault density as dependent variable were excluded from the review because we wanted to represent the entire research field. In this section, studies using fault density are compared with LOC size ranking ability to assess the LOC predictive capabilities.

To some extent, size correlates with the number of faults [130,151,154,112,83,78], but there is no strong evidence that size metrics, like LOC, are a good indicator of faults [84,59]. A strong correlation was observed between the size metric LOC and fault proneness in pre-release and small studies, while in post-release and large studies, only a weak association was found (Table 5). This may indicate that studies with lower validity (smaller data sets) gave greater significance to the LOC metric than studies with higher validity (larger data sets). Therefore, the reliability of the studies in terms of study quality assessment was taken into account and the overall effectiveness of the LOC metric was estimated as moderate.

4.3.2. RQ2.2: Are complexity metrics useful for fault prediction? Do they have additional benefits after being adjusted for size?

Popular complexity metrics, like McCabe's cyclomatic complexity [44] and Halstead's metrics [24], were used in 22 and 12 studies, respectively. McCabe's cyclomatic complexity was a good predictor of software fault proneness in [127,156,112,111,121,92,75], but not in [64,87,141,148]. Cyclomatic complexity was fairly effective in large post-release environments using OO languages but not in small pre-release environments with procedural languages (Table 5). This indicates that cyclomatic complexity may be more effective in large and OO environments. One potential explanation for this could be that in large data sets, modules are more complex than in small data sets. We assumed that modules in small data sets (usually academic and written by students) were not as complex as in large industrial cases. Also, the modules used in OO programming languages (usually class) are generally bigger than modules used in procedural languages (usually method) and may, therefore, be more complex. Although the cyclomatic complexity was not found to be effective in all the categories, the overall effectiveness was estimated as moderate, as there were individual studies reporting its usability.

Poor results were reported for Halstead's metrics, which were significant in [116,148], but not in [101,112,111]. According to the evidence gathered, Halstead's metrics were not as good as McCabe's cyclomatic complexity or LOC [111]. They were found to be somewhat effective in pre-release and small studies, suggesting low validity of the results. The total number of operators N_1 and the total number of operands N_2 performed better than the number of unique operators η_1 and the number of unique operands η_2 [148]. In other categories, Halstead's metrics were ineffective when compared to other metrics. Hence, they were estimated as inappropriate for software fault prediction.

Fenton and Ohlsson [84] reported that complexity metrics are not the best, but reasonable predictors of fault-prone modules. They found a high correlation between complexity metrics and LOC. Zhou et al. [154] also noted that class size has a strong confounding effect on associations between complexity metrics and fault-proneness, and that the explanatory power of complexity metrics, in addition to LOC, is limited. McCabe's and Halstead's metrics were highly correlated to each other and to the lines of code in [88,18]. The high correlation between complex metrics and LOC is reasonable, since complexity metrics are essentially a size measurement [112]. From the evidence gathered, it seems complexity metrics are not bad fault predictors, but others are better [112,84,88,154,155].

4.3.3. RQ2.3: Are OO metrics useful for fault prediction? Do they have additional benefits after being adjusted for size?

The most frequently used and the most successful among OO metrics were the CK metrics. However, not all CK metrics performed equally well. According to various authors [62,66,67,81,149,89,153,128,132,136,53,137] the best metrics from the CK metrics suite are CBO, WMC and RFC, which were effective across all groups (Table 5). LCOM is somewhere in between [62,89,132,137] and was found to be effective in small pre-release studies. When all the evidence is considered and compared to other CK metrics, LCOM is not very successful in finding faults. DIT and NOC were reported as untrustworthy [139,153,89,128,132,136,87,83,137]. DIT was only significant in some small and pre-release studies, while NOC was unreliable and occasionally inverse significant. This means that the module has greater fault po-

Author's personal copy

tential if it has fewer children, which is the opposite of the NOC original definition. Consequently, DIT and NOC were assessed as ineffective OO metrics.

Abreu and Carapuca MOOD [1,51] and Bansiya and Davis QMOOD metric suites [3] were only validated in a few studies. Olague et al. [128] reported that the QMOOD metrics were, while the MOOD metrics were not, suitable for predicting software fault proneness. They also stated that the CK metrics performed better than QMOOD and MOOD metrics. Among QMOOD metrics, CIS and NOM metrics performed better than others. In [82], MOOD metrics were evaluated and compared with the Martin suite [43] and CK metrics. The MOOD metrics were outperformed by the Martin suite. In the MOOD suite, the CF metric had the highest impact.

The MOOD metrics were also theoretically evaluated in [31]. This study is not included in the list of primary studies because there was no empirical validation. The study reported that the MOOD metrics operated at the system level, while the CK metrics operated at the component level. Therefore, MOOD metrics are appropriate for project managers to assess the entire system, and CK metrics are more appropriate for developers and monitoring units. Al Dallal [56,55] used CAM metrics from the QMOOD metrics suite to compare it with other cohesion metrics and the proposed SSC metric. In the two studies, both CAM and SSC performed well.

The predictive performance of cohesion metrics LCC and TCC [5] was estimated as modest by all ten studies [71,65,69,66,67,52,109,53,56,55]. They have a similar definition and share similar performance [53,56,55]. LCC was found to perform slightly better than TCC [69,52]. In [109] TCC and LCC performed the worst among ten cohesion metrics, whereas TCC was slightly better than LCC.

Briand et al. [70] studied and proposed new object-oriented metrics. The coupling metrics were reported to be successful [70,71,65,69,66,67,8,81,54,53]. The coupling metrics were further divided into import coupling and export coupling. The import coupling metrics were reported to be superior over export coupling [69,71,66–68,54,53]. Only in [81] were export coupling metrics more successful.

El Emam et al. [80] investigated the association between class size, CK metrics and Lorenz and Kidd OO metrics. They demonstrated that after controlling for size, metrics correlation with faults disappeared, indicating a strong correlation between OO metrics and size. The association between OO metrics and size was also observed in [60], where the cost-effectiveness of OO metrics was studied. The OO metrics were good predictors of faulty classes, but did not result in cost-effective prediction models. Subramanyam and Krishnan [138] tended to agree to some extent with El Emam, but they suggested that the additional effect of OO metrics beyond the one explained by size is statistically significant.

Since only a few studies have taken into account the potential correlation between size and OO metrics, additional validation is needed to assess the impact of software size on OO metrics.

4.3.4. RQ2.4: Are process metrics superior to traditional and OO metrics? Is the answer dependent upon the life cycle context?

Process metrics are computed from the software change history. They can be further divided into delta metrics and code churn metrics. Delta metrics are calculated as the difference of metrics values between two versions of a software. They show the end result, i.e. how the metrics value has changed between two versions, but not how much change has occurred. For example, if several lines of code have been added, this change will be reflected in a changed delta value. But if the same number of lines have been added and removed, the delta value will remain the same. This weakness is corrected by code churn metrics, which capture the overall change to the software between two versions. The delta and the code churn metrics can be computed for any metric [22].

The first study in our review to investigate the process metrics was [116]. Of the three independent variables of code churn, deltas and developer measures used in the study, code churn had the greatest correlation with trouble reports. There was no apparent relationship between the number of developers implementing a change and the number of trouble reports.

Graves et al. [88] looked at how code changes over time. They reported that the number of changes to code in the past, and a measure of the average age of the code, were successful predictors of faults; clearly better than product measures such as lines of code. Their most successful model, the weighted time damp model, predicted fault potential using a sum of contributions from all the changes to the module in its history, whereas large and recent changes contribute the most to fault potential. In [130,93] new and changed files had more faults than existing, unchanged files with otherwise similar characteristics.

Moser et al. [115] compared 18 process metrics to static code metrics. They reported that process related metrics contain more discriminatory and meaningful information about the fault distribution in software than the source code itself. They found four metrics to be powerful fault predictors and explained that files with a high revision number are by nature fault prone, while files that are part of large CVS commits are likely to be fault free, and bug fixing activities are likely to introduce new faults. Refactoring, meanwhile, seems to improve software quality.

In [117], relative code churn metrics were found to be good predictors of binaries' fault density, while absolute code churn was not.

Arisholm et al. [60] performed an extensive comparison of OO, delta and process metrics on 13 releases of a large Java system. They found large differences between metrics in terms of cost-effectiveness. Process metrics yielded the most cost-effective models, where models built with the delta and the OO metrics were not cost-effective. Process metrics were also the best predictor of faulty classes in terms of ROC area, followed by the OO and delta metrics. The differences between metrics were not as significant as they were in terms of cost-effectiveness. A good prediction of faulty classes and low cost-effectiveness of the OO metrics may be explained by the aforementioned association with size measures [80].

Developer metrics were investigated by several studies, but their usability in fault prediction remains an important unanswered research question. Schröter et al. [135] investigated whether specific developers are more likely to produce bugs than others. Although they observed substantial differences in failure density in files owned by different developers in pre-release and post-release, they suspect that the results do not indicate developer competency, but instead reflect the complexity of the code.

Weyuker et al. [147,146,131] introduced three developer metrics: the number of developers who modified the file during the prior release, the number of new developers who modified the file during the prior release, and the cumulative number of distinct developers who modified the file during all releases through the prior release. Only a slight improvement of prediction results was recorded when developer information was included. Graves et al. [88] found that the number of different developers, who worked on a module, and a measure of the extent to which a module is connected with other modules, did not improve predictions. On the other hand, developer data did improve predictions in [118,93,110]. All four studies have the same quality assessment score and are very similar in all properties, but different in the property 'Programming language'. Graves et al. [88] used procedural programming language, while in [118,93,110] OO programming language was used. However, we do not believe this is the reason for the contradictory results.

Source code metrics (i.e. size, complexity and OO metrics) do not perform well in finding post-release faults [57,128,136,60],

Author's personal copy

because these are more likely due to the development process and less likely due to the software design [136]. Fault prediction models are influenced not just by source code but by multiple aspects, such as time constraints, process orientation, team factors and bug-fix related features [46].

Moser et al. [115] gives a simple explanation why process metrics contain more meaningful information about fault distribution than source code metrics. They explain that a very complex file, which would be classified as faulty by a prediction model based on complexity metrics, can be fault free, because it was coded by a skilled developer who did a very prudent job. On the other hand, if a file is involved in many changes throughout its life cycle, there is a high probability that at least one of those changes will introduce a fault, regardless of its complexity.

In contrast to source code metrics, process metrics were found to be useful in predicting post-release faults [100,107,58,73,85, 91,60,93,110,88,115,117,118]. Only in [150] did additional process metrics not improve classification accuracy.

According to included studies and to our assessment of the metrics' effectiveness (Table 5), the best process metrics are code churn, the number of changes, the age of a module and the change set size. They were strongly related to fault proneness in all categories. When compared to other three metrics, the change set size metric showed slightly poorer results in studies using procedural languages. Code delta, the number of developers and the number of past faults metrics exhibited some evidence relating them to fault proneness, but the results were not as strong as with other process metrics. Code churn metrics were found to be more appropriate than delta metrics. The developer metric showed only moderate correlation with faults and seemed to dependent upon the environment, since different observations were reported. It is not immediately clear whether past faults are related to faults in the current release, but there seems to be a weak connection between them.

Industry practitioners looking for effective and reliable process metrics should consider code churn, the number of changes, the age of a module and the change set size metrics. Researchers looking for poorly investigated areas may consider the number of past faults and the change set size metrics, since there is not much evidence relating them to faults. Another challenging area would be to find new ways to measure process related information and to evaluate their relationship with fault proneness.

4.3.5. RQ2.5: Are there metrics reported to be significantly superior for software fault prediction, and if so which ones?

Although there is evidence for each metric category associating metrics with fault proneness, we have considered how strong the evidence is, taking into account contradictory arguments and studies' reliability. According to the 106 primary studies, there are metrics reported to be more successful at fault prediction than others. From the evidence gathered, it seems OO and process metrics are more successful at fault prediction than traditional size and complexity metrics.

In the OO category, the most frequently used and the most successful metrics are the CK metrics. The CBO, WMC and RFC were reported to be the best from the CK metrics suite. The coupling metrics were reported to be better than inheritance and cohesion metrics.

Process metrics were found to be successful at finding post-release faults [100,107,58,73,85,91,60,93,110,88,115,117,118], while source code metrics were not [57,128,136,60]. Metrics code churn, the number of changes, the age of a module and the change set size have the strongest correlation with post-release faults among process metrics. However, it is not immediately clear whether fault prediction is also influenced by having developer data available. Some studies, i.e. [118,93,110], found that additional developer data improved prediction results, while others [116,88,135, 147,146,131] did not find any or found just a slight improvement of prediction results, when developer information was included. Only weak evidence supporting correlation with faults was also exhibited for the past faults and code delta metrics. Therefore, metrics developer, past faults and code delta are not among the most successful process metrics.

Our findings are similar to those of Hall et al. [23] for size, complexity and OO metrics, but differ regarding process metrics. Hall reported that process metrics performed the worst among all metrics. This could be due to the quantitative model used, which only takes into account the raw performance of the models, without considering the context in which the studies were performed. Poor process metrics performance might be explained by the fact that process metrics are usually evaluated in post-release software, where faults are harder to find. In addition, Kitchenham [34] also concluded that process metrics are more likely to predict post-release faults than static code metrics.

There have been many published studies that validate CK metrics, and much attention has been paid to LOC, but there are not that many studies using process metrics. In the future, we would like to see more studies proposing and validating process metrics, since they may be of great potential in software fault prediction.

4.3.6. RQ2.6: Are there metrics reported to be inferior for software fault prediction, and if so which ones?

Complexity metrics do have some predictive capabilities, but may not be the best metrics for software fault prediction, since they are highly correlated with lines of code [84,88,112,154].

Size measures like LOC metrics are simple and easy to extract; but as with the complexity metrics, they only have limited predictive capabilities. They are partly successful in ranking the most fault prone modules, but are not the most reliable or successful metrics [84,130,59,151].

Neither are all OO metrics good fault predictors. For example, NOC and DIT are unreliable and should not be used in fault prediction models [139,153,89,128,132,136,87,83,137]. There is also little evidence for other OO metrics like the MOOD and QMOOD metrics, but they do not promise much on the other hand [128].

There are some metrics that could not be classified either as successful or unsuccessful fault predictors, due to either the lack of research (e.g. MOOD, QMOOD, individual process metrics) or conflicting results (e.g. LCOM, developer metrics). These metrics will be left for future research to determine their usefulness.

4.4. RQ3: What data sets are used for evaluating metrics?

In the case of our primary studies, data was gathered from different software items and software development processes. This section describes what data sets were used, whether they were publicly available, the size of the data sets, and what programming languages were used.

4.4.1. RQ3.1: Are the data sets publicly available?

62 (58%) of the studies used private, 22 (21%) partially public and only 22 (21%) public data sets. The number of studies using private data sets is concerning, since 58–79% of the studies are, in our opinion, not repeatable, and the results can therefore be questioned. Using private data sets lowers a study's credibility.

In 2005, the PROMISE repository [6] was introduced. It includes publicly available software measurements and is free for anyone to use. One would imagine that it would help raise the question of publicly available data sets, but only a slight improvement can be detected. After 2005, 41% of the selected studies still used private data sets, 30% used partially public and 28% used publicly available data sets.

Author's personal copy

4.4.2. RQ3.2: What are the size of the data sets?

58 (55%) studies used large and 12 (11%) used medium sized data sets. In our estimation, these two groups have data sets large enough to make a trustworthy conclusion. We are concerned with 36 (33%) studies, which used small data sets, because there is a great threat of generalizing the results. Data sets were classified as small if they used less than 200 modules (classes, files, etc.), which is a low value of cases for a predictive model to make a reliable decision. Such a model can easily be influenced by the underlying data sets and any conclusion drawn from it could be misleading. Therefore, the results of the studies with small data sets should be interpreted with caution.

After 2005, larger data sets have been used. The number of large data sets (72%) increased and the number of small (18%) and medium (10%) sized data sets decreased. Hence, with respect to the previous research question, and with respect to this research question, we can discern a positive trend.

Process metrics were almost always evaluated on large data sets (Fig. 4), which increases the validity of studies using process metrics. However, the validity of studies using OO metrics is a source of concern, since half of the studies used small data sets. Small data sets are only strongly associated with OO metrics, but not with two other metrics' categories. Because of this phenomenon, special attention was paid to the results of the studies using small data sets and OO metrics.

4.4.3. RQ3.3: What software programming languages are used to implement software from which data sets are extracted?

The mostly frequently used programming languages were C++ (35%), Java (34%) and C (16%). Other programming languages were rarely used (8% in total), while in 7% of the studies the programming language was not even stated. It is interesting that, in addition to Java, not many other modern OO programming languages were used (two studies, [121,107], used C#). In the future, more studies should use OO programming languages other than Java (e.g. C#, Ruby, Python, not to mention web programming languages such as PHP and JavaScript) to see how metrics perform in different programming languages, since the predictive capabilities of metrics may differ between programming languages [138].

The choice of the programming language is associated with the choice of software metrics. OO metrics cannot be used in procedural programming languages, because they do not include object-oriented concepts like coupling, cohesion and inheritance. Fig. 5 shows the number of studies using metrics in procedural and OO programming languages. It confirms that OO metrics are only used in OO programming languages. Traditional and process

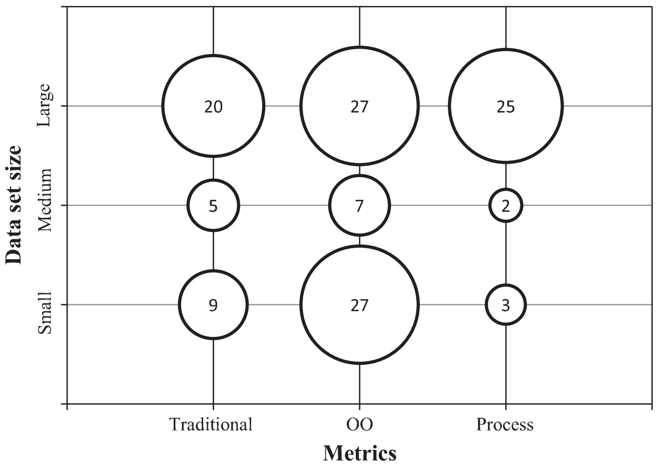


Fig. 4. The number of studies using metrics by data set sizes.

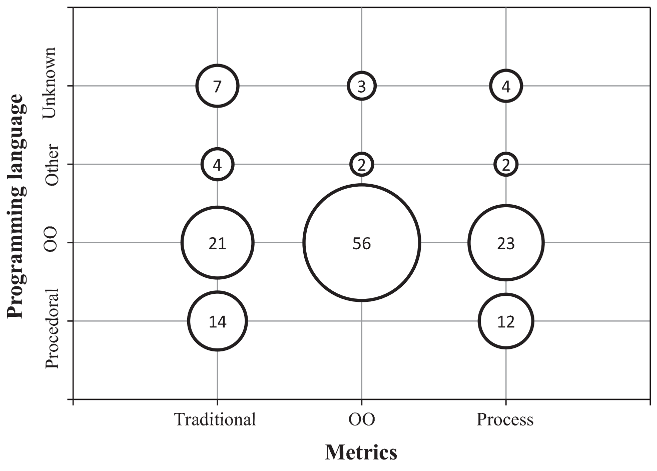


Fig. 5. The number of studies using metrics by programming languages.

metrics can be used in both types of programming languages, but they were more frequently used in OO programming languages, because of their prevalence. One would expect traditional metrics to be more often used in procedural than OO programming languages, but according to Fig. 5, the opposite is the case. This could be due to the large number of studies validating OO metrics and using traditional metrics for comparison. The more frequent use of process metrics in object-oriented than procedural programming languages is expected, since process metrics are newer.

4.5. RQ4: What are the software development life cycles in which the data sets are gathered?

The number of studies using pre-release software (64%) is larger than the number of studies using post-release software (36%). In recent years there was a slight increase in studies using post-release software. Since 2005, the number of studies using post-release software (47%) is about the same as the number of studies using pre-release software (53%).

4.5.1. RQ4.1: Are there different metrics used in different software development life cycles?

Different metrics are used in pre-release compared to post-release software. In pre-release, size, complexity and OO metrics are used, while process metrics are not used, because process-related information is normally not available and only one version of software exists. In post-release, size, complexity and OO metrics can be used, but they are rarely found as they do not predict faults very well in long, highly iterative evolutionary processes [57,128,136,60]. In post-release, process metrics are commonly used, because of their success in predicting post-release faults [88,115,73,85,60,93].

4.5.2. RQ4.2: Is there proof that fault prediction accuracy differs across various software development life cycles?

Static code metrics were found to be reasonably effective in predicting evolution changes of a system design in the short-cycled process, but they were largely ineffective in the long-cycled evolution process [57]. For highly iterative or agile systems, the metrics will not remain effective forever [128,136]. Static code metrics are, on the one hand, suitable for observing a particular state in a software iteration but, on the other hand, their prediction accuracy decreases with each iteration [128,136,57,156,89]. One reason for this might be that there are generally more faults in pre-release than in post-release [136]. Post-release faults are also harder to find, because they are less influenced by design and more by

Author's personal copy

development properties, which cannot be easily detected by static code metrics [136].

4.6. RQ5: What is the context in which the metrics were evaluated?

In this section, we present the context of the selected studies (RQ5). In Sections 4.6.1, 4.6.2, 4.6.3, 4.6.4, 4.6.5 the distribution of studies according to the properties ‘Researcher’, ‘Organization’, ‘Modeling technique’, ‘Dependent variable’ and ‘Dependent variable granularity’ are presented.

4.6.1. Researcher (P6)

The research on software metrics used in software fault prediction is driven by researchers from academia. 87% of the studies were performed by researchers from academia and only 13% by researchers from the industry. It is difficult to assess the industrial applicability of the research, when very few researchers are actually from the industry. This may also point to a certain degree of disinterest from the industry's point of view.

Academic researchers were strongly interested in OO metrics, while the majority of researchers in the industry used process metrics (Fig. 6). This could mean that process metrics are more appropriate than other categories for industrial use. We would advise academic researchers to follow researchers in the industry and focus on process metrics.

Industry experts may be interested in the following studies [149,117,130,120,124,119,128,129,147,118,146,92,131,123]. They were performed by researchers in the industry on large data sets, but unfortunately private ones. Only [128,129] used medium and partially available data sets. According to our quality assessment, they are as reliable as the studies performed by researchers from academia. Their average quality assessment score is 16.4, while the overall average is 16.3. The quality assessment score of the studies [128,129] is 19.5, hence they are highly reliable. They investigate CK, MOOD, QMOOD object-oriented metrics and McCabe's cyclomatic complexity. Less reliable studies are [147,92], with a score of 13 and 13.5.

4.6.2. Organization (P7)

Although the majority of researchers are from academia, 76% of the studies used industrial software to evaluate metrics performance. Still, 24% of the studies used academic software, which was usually small and written by students.

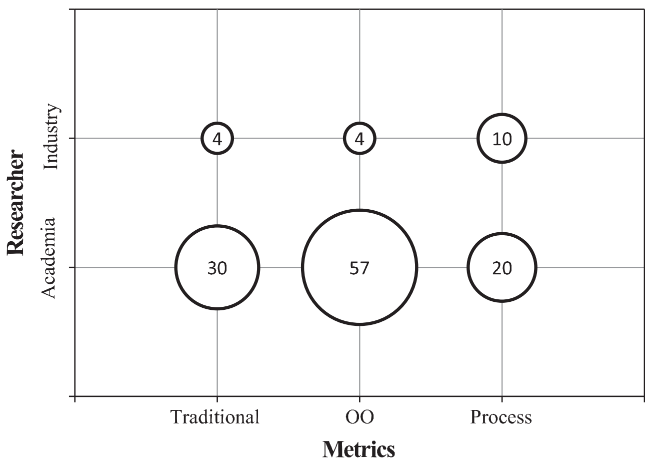


Fig. 6. The number of studies using metrics by researchers from academia and the industry.

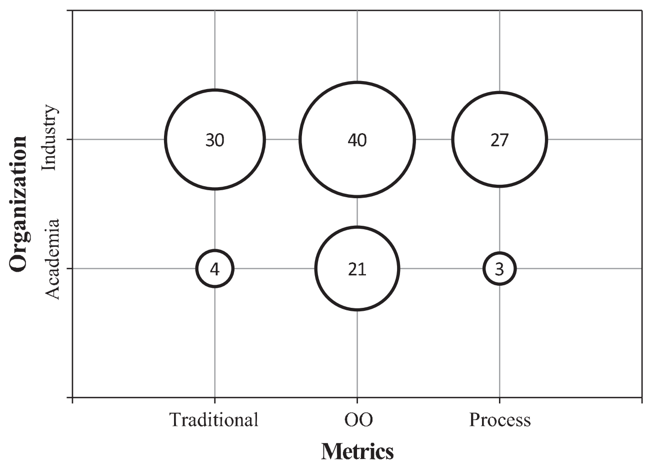


Fig. 7. The number of studies using metrics by organizations.

Almost all studies performed on academic data sets used OO metrics, while industrial data sets were equally represented by all three metrics' groups (Fig. 7). This indicates the wide acceptance of OO metrics within the academic research community. The results are in alignment with those of the property ‘Researcher’, where the majority of researchers from academia used OO metrics.

A similar pattern regarding OO metrics was observed in Fig. 4, where the use of metrics was analyzed against the data set size. Half of the studies validating OO metrics used small data sets. When comparing studies using academic software and studies using small data sets, a strong correlation was discovered. Out of 25 studies using academic software, 21 studies used small, 4 studies used medium, and none of them used large data sets. Therefore, studies using academic software usually have low external validity. To get the most relevant results, from the industry's point of view, metrics' evaluation should always be performed on large, industrial software.

4.6.3. Modeling technique (P8)

In this systematic review, we focused on selecting metrics for software fault prediction and not on selection of modeling technique. Nevertheless, we extracted data about modeling techniques also, just to see which modeling techniques were the most frequently used. Statistical analyses, like logistic and linear regression, were used in 68% of the studies, followed by machine learning algorithms (24%) and correlation analysis (8%).

4.6.4. Dependent variable (P9)

The most common dependent variable used in fault prediction models is detecting whether the particular module is faulty or not (70%). Predicting how many faults may be in a module (24%) and predicting the severity of faults (3%) offers more information. Modules having more faults or having more severe faults can be prioritized and, thus, help to allocate and save resources [137]. In spite of some other proposed techniques (3%), classifying and ranking faults remain the most used dependent variables.

4.6.5. Dependent variable granularity (P10)

The prediction made for smaller modules, such as classes or files, is more effective than the prediction made for larger modules,

Author's personal copy

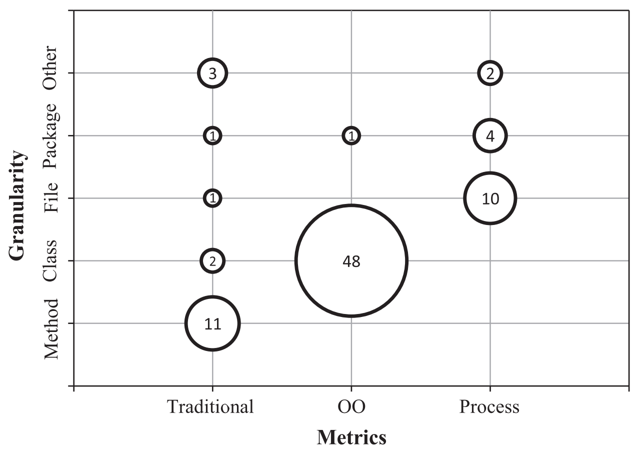


Fig. 8. The number of studies using metrics at different dependent variable granularities.

like packages, because the fault location is more precise and less work is required to review the module [76].

The granularities reported by the primary studies are class (55%), method (16%), file (13%), package (11%) and other (5%). From the selected studies, the association between metrics and dependent variable granularity can be observed (Fig. 8). Traditional metrics are associated with methods, OO metrics with classes and process metrics with files.

The above associations must be considered when choosing metrics for fault prediction model. The selection of metrics is not only related to metrics performance, but also to granularity level at which fault prediction is required. At class level, we would probably use OO metrics. At method level, traditional size and complexity metrics would be used, because OO metrics cannot be used. Process metrics are generally used at file level, because many are extracted from a version control system, which operates at file level.

5. Conclusions

The main objective of this systematic review was to find and assess primary studies validating software metrics used in software fault prediction. An extensive search was performed in order to find primary studies, which included searching seven digital libraries, snowball sampling and contacting the authors directly. Out of 13,126 studies found in digital libraries, 106 primary studies were included and evaluated according to a number of research questions. For each study, ten properties were extracted and a quality assessment was performed in order to achieve new insights.

Researchers who are new to the area can find the complete list of relevant papers in the field. They will also find the most frequently used metrics in fault prediction, with references to the studies in which they are used (Table C.7). The metrics' effectiveness is assessed to provide a clear distinction regarding their usefulness. New insights into how metrics were used in different environments were obtained through the assessment of the context. With this SLR, researchers have gained an overall overview of the area and an outline for future work. Researchers from the industry will most likely be interested in Table 5, where the effectiveness of the most frequently used metrics were assessed in a five-point scale to distinguish between metrics' usefulness and to determine the degree of effectiveness. The quality assessment of selected primary studies was used as a reliability measure. Studies with a higher quality assessment score were given greater meaning, when assessing the metrics' effectiveness. The effectiveness was assessed overall and for different environments, i.e. pre- and post- release SDLC, small and large data sets, procedural and OO programming languages. This way, researchers can distinguish which metrics were effective in a particular environment and to what extent.

For instance, researchers from the industry looking for metrics for fault prediction in large, OO, post-release systems would be advised to use process metrics, like code churn, the number of changes, the age of a module and the change set size. They could also try static code metrics (e.g. CBO, RFC and WMC), but should keep in mind that they have some limitations in highly iterative and agile development environments. Although this SLR summarizes the metrics' effectiveness, several different metrics should be tested to find the most appropriate set of metrics for each particular domain.

Object-oriented metrics (49%) were used nearly twice as often as traditional source code metrics (27%) or process metrics (24%). The most popular OO metrics were the CK metrics, which were used in almost half of the studies.

According to the selected primary studies, there are considerable differences between metrics with respect to software fault prediction accuracy. OO and process metrics were found to be more successful at fault prediction than traditional metrics.

Traditional complexity metrics are strongly associated with size measures like lines of code [84,88,112,154]. Complexity metrics and size measures have been reported to have some predictive capabilities but are not the best fault predictors in general [84,88,112,64,87,141,148,154,130,59,151].

Object-oriented metrics are reported to be better at predicting faults than complexity and size metrics [62,66,67,81,149,89, 153,128,132,136,53,137]. Although they also seem to correlate to size [80,60], they do capture some additional properties beyond size [138].

Static code metrics, like complexity, size and OO metrics, are useful for observing particular software version, but their prediction accuracy decreases with each software iteration [128,136,57,156,89]. They were found to be inappropriate for highly iterative, post-release software [57,128,136,60], where faults are mainly due to the nature of the development process and not so much to design and size properties [46,136]. In this kind of environment, process metrics are reported to perform better than static code metrics in terms of fault prediction accuracy [88,115,73,85,60,93].

The majority of the studies used size, complexity and OO metrics (70%), whereas process metrics, which have great potential, were less frequently used (23%). Researchers from the industry used mostly process metrics, whereas researchers from academia preferred OO metrics. This may indicate that process metrics are more appropriate for industrial use than static code metrics and that process metrics are what the industry wants. Therefore, we would like to make an appeal to researchers to consider including process metrics in their future studies. Other suggestions for future work are discussed below.

In most cases, one of three programming languages was used, i.e. C++ (35%), Java (34%) and C (15%), while others were rarely employed (8%). Future studies should consider other OO languages (e.g. C#) to investigate the performance of metrics in different programming languages. The metrics' effectiveness across different programming languages could be further investigated, since it appears to differ between languages [138].

Author's personal copy

Collecting software metrics and fault data can be difficult. Sometimes fault data is not available. This may be the case when a new project is started. Future studies could try to answer the question as to whether a fault prediction model can be learned on similar within company or even cross-company projects, and used on the newly started project. Turhan et al. [143] did not find cross-company models to be useful when compared to internal company models.

Validation should be performed in the most realistic environment possible in order to acquire results relevant for the industry. In realistic environments, faults are fairly random and data sets are highly unbalanced (few faults, many correct modules). The number of faults and their distribution between two releases can differ significantly. Validation techniques, like a 10-fold cross-validation, 2/3 for training and 1/3 for testing, do not take into account all the factors that real environment validation does, where fault prediction models are trained on release i and tested on release $i+1$. Some of the environment factors worth investigating may be raised with the following questions: What will happen if the model is trained on release i and used to predict faults on release $i+1$, which has several times as many faults as release i? What if there are no or only a few faults in release $i+1$? How will the model cope with a large amount of newly added functionality between two releases? What about a major refactoring? These may be difficult questions, but answers could be of great importance for practitioners in the industry. Only validation, where models are trained on release i and evaluated on release $i+1$, can determine the impact of all these, and other unpredictable factors, of the environment.

We noticed that 87% of researchers stem from academia, that 25% of the studies used academic software, that 58% of the studies used private data and that 33% used small data sets. Therefore, in the hope of improving industrial relevance, as well as the quality and validity of research, we would like to suggest that large, industrial software be used and that data sets be made publicly available whenever possible.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and suggestions. This study was partly financed by the European Union's European Social Fund.

Appendix A. Supplementary material

Extracted data from 106 primary studies and the final score of the quality assessment can be found in the online version of the article. The data is presented in tabular form and includes properties describing the study ('Title', 'Authors', and 'Year of publication'), ten extracted properties ('Metrics', 'Data set availability', 'Data set size', 'Programming language', 'SDLC', 'Researcher', 'Organization', 'Modeling technique', 'Dependent variable' and 'Dependent variable granularity') and quality assessment score ('Quality').

Appendix B. Quality checklist

Table B.6 shows the quality checklist used for the quality assessment of primary studies and results for each question. A detailed description of the quality checklist and analysis of the results are presented in Section 3.5.

Appendix C. Metrics

The most commonly used metrics in the selected studies are presented in Table C.7. The metrics were selected on the basis of the number of appearances in the studies. We excluded metrics that were only validated in the studies in which they were presented. Some metrics made it to the list in order to make a complete list of metrics belonging to the same suite and to make a clear distinction between used and unused metrics.

Table B.6
Quality checklist.

ID	Question	Yes	Partially	No
Design				
1	Are the aims (research questions) clearly stated?	106 (100.0%)	0 (0.0%)	0 (0.0%)
2	Was the sample size justified?	67 (63.2%)	12 (11.3%)	27 (25.5%)
3	If the study involves assessment of prediction model/technique, is model/technique clearly defined?	75 (70.8%)	23 (21.7%)	8 (7.5%)
4	Are the metrics used in the study adequately measured (i.e. are the metrics likely to be valid and reliable)?	102 (96.2%)	4 (3.8%)	0 (0.0%)
5	Are the metrics used in the study fully defined?	91 (85.8%)	6 (5.7%)	9 (8.5%)
6	Are the metrics used in the study the most relevant ones for answering the research questions?	102 (96.2%)	3 (2.8%)	1 (0.9%)
Conduct				
7	Are the data collection methods adequately described?	76 (71.7%)	25 (23.6%)	5 (4.7%)
Analysis				
8	Were the data sets adequately described?	56 (52.8%)	40 (37.7%)	10 (9.4%)
9	Are the statistical methods described?	55 (51.9%)	46 (43.4%)	5 (4.7%)
10	Are the statistical methods justified?	98 (92.5%)	7 (6.6%)	1 (0.9%)
11	Is the purpose of the analysis clear?	102 (96.2%)	4 (3.8%)	0 (0.0%)
12	Are scoring systems (performance evaluation metrics/techniques) described?	65 (61.3%)	39 (36.8%)	2 (1.9%)
13	Were side effects reported?	55 (51.9%)	46 (43.4%)	5 (4.7%)
Conclusion				
14	Are all study questions answered?	100 (94.3%)	6 (5.7%)	0 (0.0%)
15	Are negative findings presented?	45 (42.5%)	59 (55.7%)	2 (1.9%)
16	Are null findings interpreted? (I.e. has the possibility that sample size is too small been considered?)	46 (43.4%)	53 (50.0%)	7 (6.6%)
17	Does the study discuss how the results add to the literature?	75 (70.8%)	28 (26.4%)	3 (2.8%)
18	Does the report have implications for practice?	93 (87.7%)	13 (12.3%)	0 (0.0%)
19	Do the researchers explain the consequences of any problems with the validity/reliability of their measures?	52 (49.1%)	52(49.1%)	2 (1.9%)
20	Is the study repeatable? Are public data sets used?	22 (20.8%)	22 (20.8%)	62 (58.5%)

Author's personal copy

Table C.7
Metrics.

Author	Metric	Description	Used in
Abreu and Carapuca [1], Abreu et al. [51]	AHF	Attribute Hiding Factor	[51,128,82]
Abreu and Carapuca [1], Abreu et al. [51]	AIF	Attribute Inheritance Factor	[51,128,82]
Abreu and Carapuca [1], Abreu et al. [51]	COF	Coupling Factor	[51]
Abreu and Carapuca [1], Abreu et al. [51]	MHF	Method Hiding Factor	[51,128,82]
Abreu and Carapuca [1], Abreu et al. [51]	MIF	Method Interface Factor	[51,128,82]
Abreu and Carapuca [1], Abreu et al. [51]	POF	Polymorphism Factor	[51]
Al Dallal and Briand [56]	SCC	Similarity-based Class Cohesion	[56,55]
Bansiya and Davis [3]	ANA	Avgrage Number of Ancestors	[128]
Bansiya and Davis [3]	CAM	Cohesion Among Methods	[128,56,55]
Bansiya and Davis [3]	CIS	Class Interface Size	[128]
Bansiya and Davis [3]	DAM	Data Accesss Metric	[128]
Bansiya and Davis [3]	DCC	Direct Class Coupling	[128]
Bansiya and Davis [3]	DSC	Design size in classes	/
Bansiya and Davis [3]	MFA	Measure of Functional Abstraction	[128]
Bansiya and Davis [3]	MOA	Measure of Aggregation	[128]
Bansiya and Davis [3]	NOH	Number of hierarchies	/
Bansiya and Davis [3]	NOM	Number of Methods	[128,94,156,58,110,61,153,77,76,136]
Bansiya and Davis [3]	NOP	Number of polymorphic methods	/
Bieman and Kang [5]	LCC	Loose class cohesion	[71,65,69,66,67,52,109,53,56,55]
Bieman and Kang [5]	TCC	Tight class cohesion	[71,65,69,66,67,52,109,53,56,55]
Briand et al. [70]	ACAIC		[70,71,65,69,66,67,8,80,81,54,53]
Briand et al. [70]	ACMIC		[70,71,65,69,66,67,8,80,81,54,53]
Briand et al. [70]	AMMIC		[70,71,65,69,66,67,54,53]
Briand et al. [70]	Coh	A variation on LCOM5	[71,65,69,66,67,109,56]
Briand et al. [70]	DCAEC		[70,71,65,69,66,67,8,80,81,54,53]
Briand et al. [70]	DCMEC		[70,71,65,69,66,67,8,80,81,54,53]
Briand et al. [70]	DMMEC		[70,71,65,69,66,67,54,53]
Briand et al. [70]	FCAEC		[70,71,65,69,66,67,54,53]
Briand et al. [70]	FCMEC		[70,71,65,69,66,67,54,53]
Briand et al. [70]	FMMEC		[70,71,65,69,66,67,54,53]
Briand et al. [70]	IFCAIC		[70,71,65,69,66,67,54]
Briand et al. [70]	IFCMIC		[70,71,65,69,66,67,80,54]
Briand et al. [70]	IFMMIC		[70,71,65,69,66,67,54]
Briand et al. [70]	OCAEC		[70,71,65,69,66,67,81,54,53]
Briand et al. [70]	OCAIC		[70,71,65,69,66,67,8,81,54,53]
Briand et al. [70]	OCMEC		[70,71,65,69,66,67,8,81,54,53,83]
Briand et al. [70]	OCMIC		[70,71,65,69,66,67,8,81,54,53]
Briand et al. [70]	OMMEC		[70,71,65,69,66,67,54,53]
Briand et al. [70]	OMMIC		[70,71,65,69,66,67,54,53]
Cartwright and Shepperd [74]	ATTRIB	Attributes	[74,8]
Cartwright and Shepperd [74]	DELS	Deletes	[74,8]
Cartwright and Shepperd [74]	EVNT	Events	[74,8]
Cartwright and Shepperd [74]	READS	Reads	[74,8]
Cartwright and Shepperd [74]	RWD	Read/write/deletes	[74,8]
Cartwright and Shepperd [74]	STATES	States	[74,8]
Cartwright and Shepperd [74]	WRITES	Writes	[74,8]
Chidamber and Kemerer [12,13]	CBO	Coupling between object classes	48 studies
Chidamber and Kemerer [12,13]	DIT	Depth of inheritance tree	52 studies
Chidamber and Kemerer [12,13]	LCOM	Lack of cohesion in methods	50 studies
Chidamber and Kemerer [13], Briand et al. [66,67]	LCOM2	Lack of cohesion in methods	[69,55,109,105,71,54,53,86,56,67,66,52,65,144]
Chidamber and Kemerer [12,13]	NOC	Number of children	53 studies
Chidamber and Kemerer [13]	NTM	Number of trivial methods	[129,154]
Chidamber and Kemerer [12,13]	RFC	Response for a class	51 studies
Chidamber and Kemerer [12,13]	WMC	Weighted methods per class	44 studies

Author's personal copy

Etzkorn et al. [17], Tang et al. [139]	AMC	Average method complexity	[129,139,154,133]
Graves et al. [88]	Past faults	Number of past faults	[88,100,150,108,130,114,115,146,60,110]
Graves et al. [88]	Changes	Number of times a module has been changed	[88,100,108,120,135,119,114,115,58,73,60,61,77,93,110,76]
Graves et al. [88]	Age	Age of a module	[88,117,130,114,115,146,61,77,110,76]
Graves et al. [88]	Organization	Organization	[88,118]
Graves et al. [88]	Change set	Number of modules changed together with the module	[88,114,115,60,61,77,93,76]
Halstead [24]	N_1	Total number of operators	[116,88,101,112,111,108,113,148,96,134,142,124]
Halstead [24]	N_2	Total number of operands	[116,88,101,112,111,108,113,148,96,134,142,124]
Halstead [24]	η_1	Number of unique operators	[116,88,101,112,111,108,113,148,96,134,142,124]
Halstead [24]	η_2	Number of unique operands	[116,88,101,112,111,108,113,148,96,134,142,124]
Henderson-Sellers [25]	AID	Average inheritance depth of a class	[71,65–67,53,144]
Henderson-Sellers [25]	LCOM1	Lack of cohesion in methods	[55,105,54,53,109,56,67,71,66,52,69,65,86,144]
Henderson-Sellers [25]	LCOM5	Lack of cohesion in methods	[144,55,71,65–67,52,69,109,105]
Hitz and Montazeri [26]	Co	Connectivity	[69,71,65,109,66,67,53,56]
Hitz and Montazeri [26]	LCOM3	Lack of cohesion in methods	[55,69,53,71,109,65,67,56,66,52,144]
Hitz and Montazeri [26]	LCOM4	Lack of cohesion in methods	[55,71,69,53,66,67,109,52,65]
Lee et al. [37]	ICH	Information-flow-based cohesion	[71,65,66,69,67,86,53,144]
Lee et al. [37]	ICP	Information-flow-based coupling	[71,65,66,69,67,86,53,144]
Lee et al. [37]	IH-ICP	Information-flow-based inheritance coupling	[71,65,66,53,69,67]
Lee et al. [37]	NIH-ICP	Information-flow-based non-inheritance coupling	[66,71,65,67,69,53]
Li [41]	CMC	Class method complexity	[94]
Li [41]	CTA	Coupling through abstract data type	[136,57,94,129]
Li [41]	CTM	Coupling through message passing	[136,57,94,129,108]
Li [41]	NAC	Number of ancestor	[94]
Li [41]	NDC	Number of descendent	[94]
Li [41]	NLM	Number of local methods	[154,57,94,129,136]
[40,39]	DAC	Data abstraction coupling	[71,67,53,69,65,66,90,138]
Li and Henry [40,39]	DAC1	Data abstraction coupling	[53]
Li and Henry [40,39]	MPC	Message passing coupling	[71,69,53,67,65,66,92,138,60]
Lorenz and Kidd [42]	NCM	Number of class methods	[129,154]
Lorenz and Kidd [42]	NIM	Number of instance methods	[97,129,154]
Lorenz and Kidd [42]	NMA	Number of methods added	[67,133,53,66,71,65,80,128]
Lorenz and Kidd [42]	NMI	Number of methods inherited	[71,53,65,66,144,67]
Lorenz and Kidd [42]	NMO	Number of methods overridden	[53,67,71,66,65,80,128]
Lorenz and Kidd [42]	NOA	Number of attributes	[140,136,85,53,61,71,65–67,144,58,77,76]
Lorenz and Kidd [42]	NOAM	Number of added methods	[136]
Lorenz and Kidd [42]	NOO	Number of operations	[136]
Lorenz and Kidd [42]	NOOM	Number of overridden methods	[136]
Lorenz and Kidd [42]	NOP	Number of parents	[133,71,53,65–67]
Lorenz and Kidd [42]	NPAVG	Average number of parameters per method	[80,128]
Lorenz and Kidd [42]	SIX	Specialization index	[71,65,53,67,66,53,80]
Marcus et al. [109]	C3	Conceptual cohesion of Classes	[109,144]
McCabe [44]	CC	McCabe's Cyclomatic Complexity	[127,152,64,84,112,111,141,120,121,113,156,87,128,129,145,155,92,133,142,110,75,154,88,148]
Munson and Elbaum [116]	Delta	Code delta	[116,117,120,119,107,114,115,60,77,110,76]
Munson and Elbaum [116]	Churn	Code churn	[116,100,150,108,117,124,119,107,114,115,73,77,110,123,76]
Munson and Elbaum [116]	Change request	Change request	[116,100,60]
Munson and Elbaum [116]	Developer	Number of developers	[116,88,100,150,108,120,135,147,114,115,118,146,73,60,61,77,93,110,131,123,76]
Tegarden et al. [49] [66,67]	CLD	Class-to-leaf depth	[71,53,66,67,65]
Tegarden et al. [49] [66,67]	NOA	Number of ancestors	[53,71,67,66,65]
Tegarden et al. [49] [66,67]	NOD	Number of descendants	[63,53,71,67,66,65]
	LOC	Lines of Code	59 studies

Author's personal copy

References

[1] F.B.E. Abreu, R. Carapuca, Object-oriented software engineering: measuring and controlling the development process, in: Proceedings of the 4th International Conference on Software Quality, 1994, p. 0.

[2] A. Avižienis, J. Laprie, B. Randell, Dependability and its threats: a taxonomy, Building the Information Society (2004) 91–120.

[3] J. Bansiya, C.G. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Transactions on Software Engineering 28 (2002) 4–17.

[4] H.D. Benington, Production of large computer programs, IEEE Annals of the History of Computing 5 (1983) 350–361.

[5] J.M. Bieman, B.-K. Kang, Cohesion and reuse in an object-oriented system, ACM SIGSOFT Software Engineering Notes 20 (1995) 259–262.

[6] G. Boetticher, T. Menzies, T. Ostrand, PROMISE repository of empirical software engineering data, 2007.

[7] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of Systems and Software 80 (2007) 571–583.

[8] L.C. Briand, J. Wust, Empirical studies of quality models in object-oriented systems, in: Advances in Computers, vol. 56, 2002, pp. 97–166.

[9] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Information Sciences 179 (2009) 1040–1058.

[10] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert Systems with Applications 36 (2009) 7346–7354.

[11] C. Catal, Software fault prediction: a literature review and current trends, Expert Systems with Applications 38 (2011) 4626–4636.

[12] S.R. Chidamber, C.F. Kemerer, Towards a metrics suite for object oriented design, in: Oopsla 91 Conference Proceedings: Object-Oriented Programming Systems, Languages, and Applications, ACM, 1991.

[13] S.R. Chidamber, C.F. Kemerer, A metrics suite for object-oriented design, IEEE Transactions on Software Engineering 20 (1994) 476–493.

[14] J. Cohen, A coefficient of agreement for nominal scales, Educational and Psychological Measurement 20 (1960) 37–46.

[15] S. Counsell, P. Newson, Use of friends in C++ software: an empirical investigation, Journal of Systems 53 (2000) 15–21.

[16] K.O. Elish, M.O. Elish, Predicting defect-prone software modules using support vector machines, Journal of Systems and Software 81 (2008) 649–660.

[17] L. Etzkorn, J. Bansiya, C. Davis, Design and code complexity metrics for OO classes, Journal of Object-Oriented Programming 12 (1999) 35–40.

[18] N. Fenton, S. Pfleeger, Software Metrics: A Rigorous and Practical Approach, vol. 5, PWS Publishing Company (An International Thomson Publishing Company), 1997.

[19] F. Garcia, M. Bertoa, C. Calero, A. Vallecillo, F. Ruiz, M. Piattini, M. Genero, Towards a consistent terminology for software measurement, Information and Software Technology 48 (2006) 631–644.

[20] I. Gondra, Applying machine learning to software fault-proneness prediction, Journal of Systems and Software 81 (2008) 186–195.

[21] L.A. Goodman, Snowball sampling, The Annals of Mathematical Statistics 32 (1961) 148–170.

[22] G.a. Hall, J.C. Munson, Software evolution: code delta and code churn, Journal of Systems and Software 54 (2000) 111–118.

[23] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Transactions on Software Engineering (2011) 1–31.

[24] M.H. Halstead, Elements of Software Science, Elsevier Science Inc., New York, NY, USA, 1977.

[25] B. Henderson-Sellers, Software Metrics, Prentice-Hall, Hemel Hempstaed, UK, 1996.

[26] M. Hitz, B. Montazeri, Measuring coupling and cohesion in object-oriented systems, in: Proceedings of the International Symposium on Applied Corporate Computing, vol. 50, 1995, pp. 75–76.

[27] ISO/IEC, IEEE, ISO/IEC 12207:2008 systems and software engineering software life cycle processes, 2008.

[28] J.M. Juran, F.M. Gryna, Juran's Quality Control Handbook, McGraw-Hill, 1988.

[29] S. Kanmani, V.R. Uthariaraj, V. Sankaranarayanan, P. Thambidurai, Object-oriented software fault prediction using neural networks, Information and Software Technology 49 (2007) 483–492.

[30] T.M. Khoshgoftaar, N. Seliya, Comparative assessment of software quality classification techniques: an empirical case study, Empirical Software Engineering 9 (2004) 229–257.

[31] H.K. Kim, A Study on Evaluation of Component Metric Suites, Lecture Notes in Computer Science, vol. 3481, Springer-Verlag Berlin, Berlin, 2005. pp. 62–70.

[32] B. Kitchenham, Procedures for performing systematic reviews, Technical Report TR/SE-0401 and 0400011T.1, Keele University, UK and NICTA, AU, 2004.

[33] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical Report EBSE 2007-001, Keele University, UK, 2007.

[34] B. Kitchenham, Whats up with software metrics? A preliminary mapping study, Journal of Systems and Software 83 (2010) 37–51.

[35] A.G. Koru, K. El Emam, D.S. Zhang, H.F. Liu, D. Mathew, Theory of relative defect proneness, Empirical Software Engineering 13 (2008) 473–498.

[36] J.R. Landis, G.G. Koch, Measurement of observer agreement for categorical data, Biometrics 33 (1977) 159–174.

[37] Y.S. Lee, B.S. Liang, S.F. Wu, F.J. Wang, Measuring the coupling and cohesion of an object-oriented program based on information flow, in: Proc. International Conference on Software Quality, Maribor, Slovenia, 1995, pp. 81–90.

[38] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Transactions on Software Engineering 34 (2008) 485–496.

[39] W. Li, S. Henry, Maintenance metrics for the object oriented paradigm, in: Proceedings of First International Software Metrics Symposium, 1993, pp. 52–60.

[40] W. Li, S. Henry, Object-oriented metrics that predict maintainability, Journal of Systems and Software 23 (1993) 111–122.

[41] W. Li, Another metric suite for object-oriented programming, Journal of Systems and Software 44 (1998) 155–162.

[42] M. Lorenz, J. Kidd, Object-Oriented Software Metrics, Prentice Hall, Englewood Cliffs, NJ, 1994.

[43] R.C. Martin, Agile Software Development, Principles, Patterns, and Practices, Prentice Hall, 2002.

[44] T.J. McCabe, A Complexity Measure, IEEE Transactions on Software Engineering SE-2 (1976) 308–320.

[45] M. Petticrew, H. Roberts, Systematic reviews in the social sciences: a practical guide, European Psychologist 11 (2006) 244–245.

[46] J. Ratzinger, M. Pinzger, H. Gall, EQ-Mine: predicting short-term defects for software evolution, Fundamental Approaches to Software Engineering (2007) 12–26.

[47] J. Rosenberg, Some misconceptions about lines of code, in: METRICS '97 Proceedings of the 4th International Symposium on Software Metrics, 1997, pp. 137–142.

[48] M. Staples, M. Niazi, Experiences using systematic review guidelines, Journal of Systems and Software 80 (2007) 1425–1437.

[49] D.P. Tegarden, S.D. Sheetz, D.E. Monarchi, A software complexity model of object-oriented systems, Decision Support Systems 13 (1995) 241–262.

[50] M. Unterkalmsteiner, T. Gorschek, A. Islam, C. Cheng, R. Permadi, R. Feldt, Evaluation and measurement of software process improvement – a systematic literature review, IEEE Transactions on Software Engineering X (2011) 1–29.

Systematic review references

[51] F.B.E. Abreu, W. Melo, F. Brito e Abreu, Evaluating the impact of object-oriented design on software quality, in: Proceedings of the 3rd International Software Metrics Symposium, 1996, pp. 90–99, 189.

[52] A. Abubakar, J. AlGhamdi, M. Ahmed, Can cohesion predict fault density?, in: International Conference on Computer Systems and Applications, IEEE Computer Society, 2006, pp 889–892.

[53] K.K. Aggarwal, Y. Singh, A. Kaur, R. Malhotra, Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study, Software Process Improvement and Practice 14 (2009) 39–62.

[54] K.K. Aggarwal, Y.S. And, A.K. And, R. Malhotra, Y. Singh, A. Kaur, Investigating effect of design metrics on fault proneness in object-oriented systems, Journal of Object Technology 6 (2007) 127–141.

[55] J. Al Dallal, Improving the applicability of object-oriented class cohesion metrics, Information and Software Technology 53 (2011) 914–928.

[56] J. Al Dallal, L.C. Briand, An object-oriented high-level design-based class cohesion metric, Information and Software Technology 52 (2010) 1346–1361.

[57] M. Alshayeb, W. Li, An empirical validation of object-oriented metrics in two different iterative software processes, IEEE Transactions on Software Engineering 29 (2003) 1043–1049.

[58] M.D. Ambros, M. Lanza, R. Robbes, M. D'Ambros, On the relationship between change coupling and software defects, in: Working Conference on Reverse Engineering, 2009, pp. 135–144.

[59] C. Andersson, P. Runeson, A replicated quantitative analysis of fault distributions in complex software systems, IEEE Transactions on Software Engineering 33 (2007) 273–286.

[60] E. Arisholm, L.C. Briand, E.B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, Journal of Systems and Software 83 (2010) 2–17.

[61] A. Bacchelli, M. D'Ambros, M. Lanza, Are popular classes more defect prone?, in: Proceedings of FASE 2010 (13th International Conference on Fundamental Approaches to Software Engineering), 2010, pp. 59–73.

[62] V.R. Basili, L.C. Briand, W.L. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Transactions on Software Engineering 22 (1996) 751–761.

[63] A.B. Binkley, S.R. Schach, Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures, in: International Conference on Software Engineering, IEEE Computer Society, 1998, pp. 452–455.

[64] A.B. Binkley, S.R. Schach, Prediction of run-time failures using static product quality metrics, Software Quality Journal 7 (1998) 141–147.

[65] L.C. Briand, J. Daly, V. Porter, J. Wust, Predicting fault-prone classes with design measures in object-oriented systems, in: Proceedings of the International Symposium on Software Reliability Engineering (ISSRE 1998), 1998, pp. 334–343.

[66] L.C. Briand, J. Wust, J.W. Daly, D.V. Porter, Exploring the relationships between design measures and software quality in object-oriented systems, Journal of Systems and Software 51 (2000) 245–273.

Author's personal copy

[67] L.L.C. Briand, J. Wüst, H. Lounis, W. Jurgen, Replicated case studies for investigating quality factors in object-oriented designs, Empirical Software Engineering 6 (2001) 11–58.

[68] L. Briand, W. Melo, J. Wust, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE Transactions on Software Engineering 28 (2002) 706–720.

[69] L. Briand, J. Wüst, S. Ikonomovski, H. Lounis, Investigating quality factors in object-oriented designs: an industrial case study, in: International Conference on Software Engineering, 1999.

[70] L. Briand, P. Devanbu, W. Melo, An investigation into coupling measures for C++, in: International Conference on Software Engineering, Association for Computing Machinery, 1997, pp. 412–421.

[71] L.C. Briand, J. Daly, V. Porter, J. Wuest, Comprehensive empirical validation of design measures for object-oriented systems, in: METRICS '98, Proceedings of the 5th International Symposium on Software Metrics, 1998, pp. 246–257.

[72] R. Burrows, F.C. Ferrari, A. Garcia, F. Taiani, An empirical evaluation of coupling metrics on aspect-oriented programs, in: Proceedings International Conference on Software Engineering, 2010, pp. 53–58.

[73] B. Caglayan, A. Bener, S. Koch, Merits of using repository metrics in defect prediction for open source projects, in: 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009, pp. 31–36.

[74] M. Cartwright, M. Shepperd, An empirical investigation of an object-oriented software system, IEEE Transactions on Software Engineering 26 (2000) 786–796.

[75] I. Chowdhury, M. Zulkernine, Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities, Journal of Systems Architecture 57 (2011) 294–313.

[76] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empirical Software Engineering (2011) 1–47.

[77] M. D'Ambros, M. Lanza, An extensive comparison of bug prediction approaches, in: IEEE Working Conference on Mining Software Repositories (MSR 2010), 2010, pp. 31–41.

[78] G. Denaro, L. Lavazza, M. Pezze, An empirical evaluation of object oriented metrics in industrial setting, in: The 5th CaberNet Plenary Workshop, 2003.

[79] M. Eaddy, T. Zimmermann, K.D. Sherwood, V. Garg, G.C. Murphy, N. Nagappan, A.V. Aho, Do crosscutting concerns cause defects?, IEEE Transactions on Software Engineering 34 (2008) 497–515

[80] K.E. El Emam, S. Benlarbi, N. Goel, S.N. Rai, The confounding effect of class size on the validity of object-oriented metrics, IEEE Transactions on Software Engineering 27 (2001) 630–650.

[81] K. El Emam, W. Melo, J.C. Machado, K. El, The prediction of faulty classes using object-oriented design metrics, Journal of Systems and Software 56 (2001) 63–75.

[82] M.O. Elish, A.H. Al-Yafei, M. Al-Mulhem, Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: a case study of Eclipse, Advances in Engineering Software 42 (2011) 852–859.

[83] M. English, C. Exton, I. Rigon, B. Cleary, Fault detection and prediction in an open-source software project, in: Proceedings of the 5th International Conference on Predictor Models in Software Engineering PROMISE 09, 2009, pp. 17:1–17:11.

[84] N.E. Fenton, N. Ohlsson, Quantitative analysis of faults and failures in a complex software system, IEEE Transactions on Software Engineering 26 (2000) 797–814.

[85] J. Ferzund, S.N. Ahsan, F. Wotawa, Empirical Evaluation of Hunk Metrics as Bug Predictors, Lecture Notes in Computer Science, vol. 5891, Springer, 2009. pp. 242–254.

[86] F. Fioravanti, P. Nesi, A study on fault-proneness detection of object-oriented systems, in: Proceedings Fifth European Conference on Software Maintenance and Reengineering, 2001, pp. 121–130.

[87] B. Goel, Y. Singh, Empirical Investigation of Metrics for Fault Prediction on Object-Oriented Software, Studies in Computational Intelligence, vol. 131, Canadian Center of Science and Education, 2008. pp. 255–265.

[88] T.L. Graves, A.F. Karr, J.S. Marron, H. Siy, Predicting fault incidence using software change history, IEEE Transactions on Software Engineering 26 (2000) 653–661.

[89] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, IEEE Transactions on Software Engineering 31 (2005) 897–910.

[90] R. Harrison, S. Counsell, R. Nithi, Coupling metrics for object-oriented design, in: Metrics 1998: Proceedings of the Fifth International Software Metrics Symposium, 1998, pp. 150–157.

[91] A.E. Hassan, Predicting faults using the complexity of code changes, in: International Conference on Software Engineering, IEEE Computer Society, 2009, pp. 78–88.

[92] T. Holschuh, T. Zimmermann, M. Pauser, R. Premraj, K. Herzig, A. Zeller, S.Q.S. Ag, P. Markus, Predicting defects in SAP java code: an experience report, in: International Conference on Software Engineering Companion Volume, 2009, pp. 172–181.

[93] T. Illes-Seifert, B. Paech, Exploring the relationship of a file's history and its fault-proneness: an empirical method and its application to open source programs, Information and Software Technology 52 (2010) 539–558.

[94] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, G. Succi, Identification of defect-prone classes in telecommunication software systems using design metrics, Information Sciences 176 (2006) 3711–3734.

[95] Y. Jiang, B. Cukic, T. Menzies, Fault prediction using early lifecycle data, in: The 18th IEEE International Symposium on Software Reliability (ISSRE '07), 2007, pp. 237–246.

[96] Y. Jiang, B. Cukic, T. Menzies, N. Bartlow, B. Cuki, Comparing design and code metrics for software quality prediction, in: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering PROMISE 08, 2008, pp. 11–18.

[97] T. Kamiya, S. Kusumoto, K. Inoue, Prediction of fault-proneness at early phase in object-oriented development, in: Proceedings 2nd IEEE International Symposium on ObjectOriented RealTime Distributed Computing ISORC99, 1999, pp. 253–258.

[98] T. Kamiya, S. Kusumoto, K. Inoue, Y. Mohri, Empirical evaluation of reuse sensitiveness of complexity metrics, Information and Software Technology 41 (1999) 297–305.

[99] A. Kaur, A.S. Brar, P.S. Sandhu, An empirical approach for software fault prediction, International Conference on Industrial and Information Systems (ICIIS) (2010) 261–265.

[100] T.M. Khoshgoftaar, R.Q. Shan, E.B. Allen, Using product, process, and execution metrics to predict fault-prone software modules with classification trees, in: Proceedings of the Fifth IEEE International Symposium on High Assurance Systems Engineering, IEEE Computer Society, 2000.

[101] T.M. Khoshgoftaar, E.B. Allen, Empirical assessment of a software metric: the information content of operators, Software Quality Journal 9 (2001) 99–112.

[102] B.A. Kitchenham, L.M. Pickard, S.J. Linkman, Evaluation of some design metrics, Software Engineering Journal 5 (1990) 50–58.

[103] A.G. Koru, H. Liu, An investigation of the effect of module size on defect prediction using static measures, in: Proceedings of the 2005 workshop on Predictor Models in Software Engineering – PROMISE '05, 2005, pp. 1–5.

[104] A.G. Koru, D.S. Zhang, K. El Emam, H.F. Liu, An investigation into the functional form of the size-defect relationship for software modules, IEEE Transactions on Software Engineering 35 (2009) 293–304.

[105] S. Kpodjedo, F. Ricca, P. Galinier, Y.G. Gueheneuc, G. Antoniol, Design evolution metrics for defect prediction in object oriented systems, Empirical Software Engineering 16 (2011) 141–175.

[106] S. Kpodjedo, F. Ricca, G. Antoniol, P. Galinier, Evolution and search based metrics to improve defects prediction, in: Proceedings of the 2009 1st International Symposium on Search Based Software Engineering, 2009, pp. 23–32.

[107] L. Layman, G. Kudrjavets, N. Nagappan, Iterative identification of fault-prone binaries using in-process metrics, in: Proceedings of the Second ACM–IEEE International Symposium on Empirical Software Engineering and Measurement – ESEM '08, ACM Press, New York, New York, USA, 2008, p. 206.

[108] P. Li, J. Herbsleb, M. Shaw, Finding predictors of field defects for open source software systems in commonly available data sources: a case study of openbsd, in: 11th IEEE International Symposium Software Metrics, 2005, IEEE, 2005, p. 10.

[109] A. Marcus, D. Poshyvanyk, R. Ferenc, Using the conceptual cohesion of classes for fault prediction in object-oriented systems, IEEE Transactions on Software Engineering 34 (2008) 287–300.

[110] S. Matsumoto, Y. Kamei, A. Monden, K.-I. Matsumoto, M. Nakamura, An analysis of developer metrics for fault prediction, in: PROMISE '10 Proceedings of the 6th International Conference on Predictive Models in Software Engineering, 2010, p. 1.

[111] T. Menzies, J. Di Stefano, How good is your blind spot sampling policy?, in: Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004, pp. 129–138.

[112] T. Menzies, J.S. Di Stefano, M. Chapman, K. McGill, Metrics that matter, in: Proceedings of the 27th Annual Nasa Goddard/IEEE Software Engineering Workshop, IEEE Computer Society, 2003.

[113] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (2007) 2–13.

[114] R. Moser, W. Pedrycz, G. Succi, Analysis of the reliability of a subset of change metrics for defect prediction, in: Esem'08: Proceedings of the 2008 ACM–IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2008.

[115] R. Moser, W. Pedrycz, G. Succi, Acm, A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction, ICSE'08 Proceedings of the Thirtieth International Conference on Software Engineering, ACM, 2008.

[116] J.C. Munson, S.G. Elbaum, Code churn: a measure for estimating the impact of code change, in: Proceedings – IEEE International Conference on Software Maintenance, IEEE Computer Society, Los Alamitos, 1998, pp. 24–31.

[117] N. Nagappan, T. Ball, Use of relative code churn measures to predict system defect density, in: International Conference on Software Engineering, IEEE Computer Society, 2005, pp. 284–292.

[118] N. Nagappan, B. Murphy, V.R. Basili, Acm, The Influence of Organizational Structure on Software Quality: An Empirical Case Study, ICSE'08 Proceedings of the Thirtieth International Conference on Software Engineering, ACM, 2008.

[119] N. Nagappan, T. Ball, Using software dependencies and churn metrics to predict field failures: an empirical case study, in: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), 2007, pp. 364–373.

[120] N. Nagappan, T. Ball, B. Murphy, Using historical in-process and product metrics for early estimation of software failures, in: 2006 17th International Symposium on Software Reliability Engineering, 2006, pp. 62–74.

Author's personal copy

[121] N. Nagappan, T. Ball, A. Zeller, Mining metrics to predict component failures, in: Proceeding of the 28th International Conference on Software Engineering – ICSE '06, 2006, p. 452.
[122] N. Nagappan, L. Williams, M. Vouk, J. Osborne, Early estimation of software quality using in-process testing metrics: a controlled case study, in: Proceedings of the Third Workshop on Software Quality, ACM, 2005, pp. 1–7.
[123] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, B. Murphy, Change bursts as defect predictors, in: Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE), 2010, pp. 309–318.
[124] A. Nikora, J. Munson, Building high-quality software fault predictors, Software: Practice and Experience 36 (2006) 949–969.
[125] A. Nugroho, M.R.V. Chaudron, E. Arisholm, Assessing UML design metrics for predicting fault-prone classes in a Java system, in: IEEE Working Conference on Mining Software Repositories, 2010, pp. 21–30.
[126] A. Nugroho, B. Flaton, M. Chaudron, Empirical analysis of the relation between level of detail in UML models and defect density, in: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems MODELS LNCS, 2008, pp. 600–614.
[127] N. Ohlsson, H. Alberg, Predicting fault-prone software modules in telephone switches, IEEE Transactions on Software Engineering 22 (1996) 886–894.
[128] H.M. Olague, L.H. Etzkorn, S. Gholston, S. Quattlebaum, Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes, IEEE Transactions on Software Engineering 33 (2007) 402–419.
[129] H.M. Olague, L.H. Etzkorn, S.L. Messimer, H.S. Delugach, An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study, Journal of Software Maintenance and Evolution-Research and Practice 20 (2008) 171–197.
[130] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Predicting the location and number of faults in large software systems, IEEE Transactions on Software Engineering 31 (2005) 340–355.
[131] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Programmer-based fault prediction, in: PROMISE '10 Proceedings of the 6th International Conference on Predictive Models in Software Engineering, 2010, p. 1.
[132] G. Pai, B. Dugan, Empirical analysis of software fault content and fault proneness using Bayesian methods, IEEE Transactions on Software Engineering 33 (2007) 675–686.
[133] T.-S. Quah, Estimating software readiness using predictive models, Information Sciences 179 (2009) 430–445.
[134] Z.A. Rana, S. Shamail, M.M. Awais, Ineffectiveness of use of software science metrics as predictors of defects in object oriented software, Proceedings of 2009 Wri World Congress on Software Engineering, vol. 4, IEEE Computer Society, 2009.
[135] A. Schröter, T. Zimmermann, R. Premraj, A. Zeller, If your bug database could talk, Proceedings of the 5th International Symposium on Empirical Software Engineering, vol. 2, Citeseer, 2006, pp. 18–20.
[136] R. Shatnawi, W. Li, The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process, Journal of Systems and Software 81 (2008) 1868–1882.
[137] Y. Singh, A. Kaur, R. Malhotra, Empirical validation of object-oriented metrics for predicting fault proneness models, Software Quality Journal 18 (2010) 3–35.
[138] R. Subramanyam, M.S. Krishnan, Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects, IEEE Transactions on Software Engineering 29 (2003) 297–310.
[139] M.H. Tang, M.H. Kao, M.H. Chen, An empirical study on object-oriented metrics, in: Proceedings of the Sixth International Software Metrics Symposium, IEEE, 1999, pp. 242–249.
[140] M. Thongmak, P. Muenchaisri, Predicting faulty classes using design metrics with discriminant analysis, in: Serp'03: Proceedings of the International Conference on Software Engineering Research and Practice, vols. 1 and 2, 2003.
[141] P. Tomaszewski, L. Lundberg, H.K. Grahn, The accuracy of early fault prediction in modified code, in: Fifth Conference on Software Engineering Research and Practice in Sweden (SERPS), 2005, p. 0.
[142] A. Tosun, B. Turhan, A. Bener, Validation of network measures as indicators of defective modules in software systems, in: Proceedings of the 5th International Conference on Predictor Models in Software Engineering – PROMISE '09, 2009, p. 1.
[143] B. Turhan, T. Menzies, A.B. Bener, J. Di Stefano, On the relative value of cross-company and within-company data for defect prediction, Empirical Software Engineering 14 (2009) 540–578.
[144] B. Újházi, R. Ferenc, D. Poshyvanyk, T. Gyimóthy, B. Ujhazi, T. Gyimothy, New conceptual coupling and cohesion metrics for object-oriented systems, in: Source Code Analysis and Manipulation SCAM 2010 10th IEEE Working Conference, 2010, pp. 33–42.
[145] D. Wahyudin, A. Schatten, D. Winkler, A.M. Tjoa, S. Biffl, Defect prediction using combined product and project metrics a case study from the open source "Apache" MyFaces project family, in: Proceedings of the 34th Euromicro Conference on Software Engineering and Advanced Applications, IEEE Computer Society, 2008.
[146] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models, Empirical Software Engineering 13 (2008) 539–559.
[147] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Using developer information as a factor for fault prediction, in: Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), 2007, pp. 8–8.
[148] Z. Xu, X. Zheng, P. Guo, Empirically validating software metrics for risk prediction based on intelligent methods, Journal of Digital Information Management 5 (2007) 99–106.
[149] P. Yu, H. Muller, T. Systa, Predicting fault-proneness using OO metrics: an industrial case study, in: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, 2002, pp. 99–107.
[150] X. Yuan, T. Khoshgoftaar, E. Allen, K. Ganesan, An application of fuzzy clustering to software quality prediction, in: Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2000, pp. 85–90.
[151] H.Y. Zhang, An investigation of the relationships between lines of code and defects, in: Proceedings – IEEE International Conference on Software Maintenance, IEEE Computer Society, 2009, pp. 274–283.
[152] M. Zhao, C. Wohlin, N. Ohlsson, M. Xie, A comparison between software design and code metrics for the prediction of software fault content, Information and Software Technology 40 (1998) 801–809.
[153] Y.M. Zhou, H.T. Leung, Empirical analysis of object-oriented design metrics for predicting high and low severity faults, IEEE Transactions on Software Engineering 32 (2006) 771–789.
[154] Y.M. Zhou, B. Xu, H. Leung, B.W. Xua, On the ability of complexity metrics to predict fault-prone classes in object-oriented systems, Journal of Systems and Software 83 (2010) 660–674.
[155] T. Zimmermann, N. Nagappan, Predicting defects using network analysis on dependency graphs, in: Proceedings of the 13th International Conference on Software Engineering – ICSE '08, 2008, p. 531.
[156] T. Zimmermann, R. Premraj, A. Zeller, Predicting Defects for Eclipse, in: Third International Workshop on Predictor Models in Software Engineering PROMISE07 ICSE Workshops 2007, 2007, p. 9.