ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 6, No. 2, 161-174 (2021)
www.astesj.com
Special Issue on Multidisciplinary Sciences and Engineering

ASTES Journal
ISSN: 2415-6698

A Large Empirical Study on Automatically Classifying Software Maintainability Concerns from Issue Summaries

Celia Chen[*,1], Michael Shoga[2]

[1]*Department of Computer Science, Occidental College, Los Angeles, 90041, United States of America*

[2]*Center for Systems and Software Engineering, Department of Computer Science, University of Southern California, Los Angeles, 90089, United States of America*

ARTICLE INFO

Article history:
Received: 22 December, 2020
Accepted: 19 February, 2021
Online: 10 March, 2021

Keywords:
Software Maintainability
Open Source Software
Empirical Study
Linguistic Analysis

ABSTRACT

Software maintenance contributes the majority of software system life cycle costs. However, existing approaches with automated code analysis are limited by accuracy and scope. Using human-assessed methods or implementing quality standards are more comprehensive alternatives, but they are much more costly for smaller organizations, especially in open-source software projects. Instead, bugs are generally used to assess software quality, such as using bug fixing time as an estimate of maintenance effort. Although associated bug reports contain useful information that describe software faults, the content of these bug reports are rarely used. In this paper, we incorporate quality standards with natural language processing techniques to provide insight into software maintainability using the content of bug reports and feature requests. These issues are classified with an automated approach into various maintainability concerns whose generalizability has been validated against over 6000 issue summaries extracted from nine open source projects in previous works. Using this approach, we perform a large empirical study of 229,329 issue summaries from 61 different projects. We evaluate the differences in expressed maintainability concerns between domains, ecosystems, and types of issues. We have found differences in relative proportions across ecosystem, domain and issue severity. Further, we evaluate the evolution of maintainability across several versions in a case study of Apache Tomcat, identifying some trends within different versions and over time. In summary, our contributions include a refinement of definitions from the original empirical study on maintainability related issues, an automated approach and associated rules for identifying maintainability related quality concerns, identification of trends in the characteristics of maintainability related issue summaries through a large-scale empirical study across two major open source ecosystems, and a case study on changes in maintainability over versions in Apache Tomcat.

1 Introduction

Software maintainability measurements provide organizations with a greater understanding of how difficult it is to repair or enhance their software. The importance of having this understanding is underscored in [1], which reported that 75-90% of business and command&control software and 50-80% of cyber-physical system software costs are incurred during maintenance. In addition, maintainability serves as a crucial link to other quality characteristics. In [2], the author lists maintainability as a contributing quality to life cycle efficiency, changeability and dependability. In [3], the maintainability is a key quality in understanding software quality interrelationships. Thus, having comprehensive knowledge of software maintainability is significant in the software development and maintenance process.

A number of metrics and approaches have been developed to provide ways to measure and evaluate software maintainability. In this study, they are classified into the following categories:

- Automated analysis: Automatic analysis involves analyzing source code or other software artifacts and quantifying software maintainability into numeric results. This includes static code analysis such as measuring Maintainability Index, technical debt, code smells, and other Object-Oriented metrics [4, 5], as well as bug-focused metrics such as bug fixing time [6] and accumulated defect density [7].

[*]Corresponding Author: Celia Chen, Email: qchen2@oxy.edu

- Human-assessed analysis: Human-assessed analysis includes reuse cost models that estimate maintainability of potentially reusable components based on human-assessed maintainability aspects, such as code understandability, documentation, and developer self-reported surveys [8, 9].
- Software ontology, standards and frameworks: These introduce immense high-level knowledge, which are mostly coming from consensus wisdom, professional discipline and expert sources. This includes standards such as the ISO/IEC 25010 and Software Engineering Body of Knowledge.

While the automated analysis metrics are easy to use and often require relatively low human effort, in [10] author points out that the effective use of accuracy measures for these metrics has not been observed and there is a need to further validate maintainability prediction models. Moreover, despite having the advantage of identifying the particular parts of the software most needing maintainability improvement at the module and method level, they do not provide an overall quality status for the current version of the software.

Although bug fixing times may reflect maintenance effort [6], these bug-focused metrics also do not provide a systematic understanding of software maintainability. Furthermore, these metrics do not utilize the information provided by the natural language descriptions due to their unstructured nature.

On the other hand, human-assessed analyses are able to more accurately reflect maintenance effort, yet they are limited in use due to cost and subjectivity based on developers' skills and experience [8, 11].

Software ontology, standards and frameworks tend to be used in larger organizations as guidelines during the development process. They provide insightful knowledge in understanding, evaluating, and improving a system's maintainability planning, staffing, and preparation of technology for cost-effective maintenance. However, it is very difficult to enforce standards on actual program behavior. Moreover, while standardizing the process can help make sure that no steps are skipped, standardizing to an inappropriate process can reduce productivity, and thus leave less time for quality assurance. Especially in smaller organizations and open source ecosystems, it is extremely difficult to apply and enforce these paradigms due to their limited resources and functionality-focused nature.

To provide a way to effectively measure and keep track of the overall maintainability while involving relatively low human effort, we utilize bug report information in conjunction with a software maintainability ontology to assess software maintainability at the system level in an initial empirical study [12]. By manually mapping over 6000 bug reports to maintainability subgroup software qualities (SQs) in the ontology, we validated the approach to evaluate overall system maintainability. However, this approach is limited by the amount of manual effort needed for mapping the bug reports. To overcome the high effort requirements, we incorporate natural language processing techniques to automatically classify "issue summaries," which include the descriptions of bug reports and feature requests, to the maintainability subgroup SQs. In this paper, we provide a refinement of definitions from the original empirical study on maintainability related issues and the rule set. We expand upon the scale of the analysis done in [12], made possible by the fuzzy classifier, to identify trends in maintainability related issue summaries from two major open-source software ecosystems. We further perform an in-depth case study on the maintainability changes over versions and time in Apache Tomcat. In total we classify 229,329 issue summaries from 61 projects and the trends over 7 versions and 20 years in Apache Tomcat.

The rest of this paper is organized as follows. Section 2 summarizes related work and presents the differences of those compared to our study. Section 3 describes the background of the automated approach and introduces the research questions and design of an empirical study on maintainability trends in two open-source software ecosystems. Section 4 discusses the results, analysis and implications. Section 5 concludes the study.

2 Related Work

2.1 Software Maintainability Measurement

Maintainability Index (MI) is the most widely used metric to quantify maintainability in software projects. Since its introduction in 1992 [13], several variations have been developed [14, 15]. While it is widely used, the metric's effectiveness has been brought into question and several shortcomings identified [16].

Other approaches to measuring maintainability have incorporated other metrics as well as frameworks and ontology. In [17], the author provided an overview of an approach that uses a standardized measurement model based on the ISO/IEC 9126 definition of maintainability and source code metrics. These metrics include volume, redundancy, complexity and more.

In [18], the author investigated 11 different types of source code metrics in an empirical study to develop a maintainability prediction model for Service-Oriented software and compare their model with the Multivariate Linear Regression (MLR) and Support Vector Machine (SVM) approaches. They found that using a smaller set of source code metrics performed better than when they used all of the available metrics.

Approaches utilizing machine learning have also been proposed. In [19], the author conducted a comparative study on using machine learning algorithms for predicting software maintainability on two commercial ADA datasets. They examined Group Method of Data Handling, Genetic Algorithms, and Probabilistic Neural Network with Gaussian activation function for predicting a surrogate maintenance effort measure, the number of lines of code changed per class over a three year maintenance period. Their results showed improvement over previously reported models.

In [20], the author proposed an LSTM algorithm for software maintainability metrics prediction. They considered 29 OO metrics and applied their approach on a large number of open source projects. In addition to comparing against other machine learning algorithms, they also used FSS to determine which metrics are most relevant for maintainability prediction.

In [21], the author presented a study using several classifiers to evaluate maintainability at the class level using the output of different static analysis tools. In their approach, ConQAT, Teamscale, and Sonarqube are used to extract metrics such as SLOC, average method length, clone coverage, etc. The classifiers are trained using

expert-labeled data from three different systems. Their best results provided a classification accuracy of 81% and a precision of 80%.

2.2 Bug Characteristics Analysis with Natural Language Processing

Several studies have investigated the characteristics of bugs and bug reports through the use of natural language processing.

In [22], the author collected 709 bugs including security related and concurrency bugs. They analyzed the characteristics of those bugs in terms of root causes, impacts and software components. Their findings reveal characteristics of memory bugs, semantic bugs, security bugs, GUI bugs, and concurrency bugs. They verified their analysis results on the automatic classification results by using text classification and information retrieval techniques.

In [23], the author proposed an approach to binary classification of bug reports into 'bug' and 'nonbug' by leveraging text mining and data mining techniques. Analyzing the summary and some structured features including severity, priority, component, and operating systems, they use Bayesian Net Classifier as the machine learner. They performed an empirical study of 10 open source projects to validate their method and provide a MyLyn plugin prototype system that will classify given reports.

In [24], the author analyzed bug reports from nine systems and found that a large percentage of bug reports lack Steps to Reproduce (S2R) and Expected Behavior (EB) information. They in turn developed an automated approach to detect missing S2R and EB from bug reports. They produced three versions using regular expressions, heuristics and natural language processing, and machine learning. They found their machine learning version to be the most accurate with respect to F1 score, but the regular expressions and heuristics and natural language processing approaches had similar accuracy results without training.

In [25], the author constructed models for identifying security and performance related bug reports utilizing feature selection, random under-sampling, and Naive Bayes Multinominal approach. They evaluated their approach on datasets of bug reports from four software projects, achieving average AUC values of 0.67 and 0.71 for their security and performance models respectively.

Summing up, here is how our work differs from the existing studies: with regard to measurement of maintainability, our work enables study of maintainability evolution with relatively low cost. By using issues, preexisting software artifacts, it allows for expert knowledge to be applied to open source software systems wherein there is less control over development and maintenance tasks.

3 Empirical Study

3.1 Background

This section presents the software maintainability ontology used and an extension of the SQ definitions provided in [12]. It also provides a brief summary on our previous works and the overall automated approach.

3.1.1 Software Maintainability Ontology Background

The ontology provided in [2] presents maintainability as depending on two alternative SQs, repairability and modifiability, which handle defects and changes respectively. These SQs are further enabled by several subgroups. The automated approach focuses on maintainability in the context of these mean-ends SQs as shown in Figure 1.

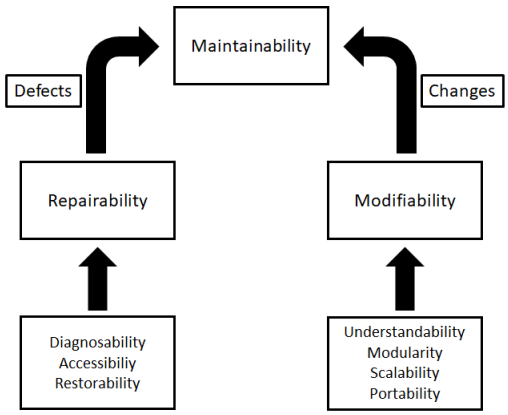


Figure 1: Software maintainability ontology hierarchy

The following are the refined definitions for each subgroup SQ to better capture the scope of these quality concerns.

Repairability involves handling of defects in software. It is enabled by the following SQs:

- ***Diagnosability***:

  Diagnosability is the characteristic of being diagnosable. It is the property of a partially observable system with a given set of potential faults, which can be detected with the certainty given finite observation. Issues that affect this SQ involve problems with lack of logging and diagnosability management, faulty error messages and the process of tracing where they originate, failure of tests, and insufficient information provided for accurate assessments [26]–[28].

- ***Accessibility***:

  Accessibility [29] generally describes the ability of a software system to accommodate people with special needs. This requires a software system to be suitable for most of the potential users without any modifications and be easily adaptable to different users with adaptable and customized user interfaces.

  Another definition for accessibility is at the architecture level. The JCIDS manual [30] defines the Accessibility of Architectures as the ability to grant access to authorized users in a timely fashion in order to "support architecture-based analysis and decision making processes." In this paper, accessibility is defined as the quality of being available and reachable, which involves whether the intended areas of a software system can be accessed as desired. Issues that affect this SQ prevent authorized users from accessing data or functions due to things such as redirects to unintended locations, broken links to intended areas, and incorrect user permission and authorization.

- ***Restorability***:

  Restorability describes the ability of a software system to restore to a previous state. Issues that affect this SQ include activities such as clearing of caches, refreshing settings, proper removal of data and backups of the current system.

Modifiability involves handling of software changes. It is enabled by the following SQs:

- ***Understandability***:

  Software understandability can be considered in the context of source code as well as non-source code artifacts and further depends on the person assessing the software. This may include the level of experience and familiarity with the software's code base if considering a developer's perspective or whether or not the software is clear in its usage and applicability if considering an end user's perspective. Understandability can have an impact on maintenance tasks especially in cases where the original developers are not the ones responsible for maintaining the system. Further explanation of software understandability is provided in [31].

  Issues that affect this SQ involve activities such as system enhancement, lack of explanations and comments, confusing or inaccurate descriptions, presence of deprecated software and more.

- ***Modularity***:

  Modularity involves separation of code into modules. It indicates the degree to which a system's components are made up of relatively independent components or parts which can be combined [32, 33].

  Issues that affect this SQ involve unwanted interactions between different modules and separation of one module into multiple modules.

- ***Scalability***:

  Scalability is the ability of a system to continue to meet its response time or throughput objectives as the demand for the software functions increases [34, 35]. Issues that affect this SQ involve latency in functionality, hangs, and insufficient resources for functionality to scale up or down.

- ***Portability***:

  Portability refers to the ability of a software unit to be ported to a given environment and being independent of hardware, OS, middle-ware, and databases [36, 37]. Issues that affect this SQ prevent proper interfacing between software components and external platforms.

3.1.2 Background Studies

In our previous empirical study [12], we manually analyzed 6372 bugs found in the Mozilla community. By categorizing them into one of the subgroup SQs described above, we identified various trends in maintainability changes as software evolves and the relationships between these subgroup SQs. The findings were valuable but it was difficult to scale up the study due to the large amount of manual effort required to produce such mappings between bug reports and subgroup SQs.

Thus, a manual analysis on the ground-truth dataset[1] was first performed, and we identified three types of linguistic patterns from bug reports: lexical patterns, syntax patterns and semantic patterns. These patterns illustrate the recurrent linguistic rules that users are likely to use when reporting bugs or requesting new features. Motivated by these heuristic linguistic patterns, we proposed a fuzzy classifier [31, 38] that aims to identify the maintainability subgroup SQ concerns expressed in issue summaries. Based on the definitions of these patterns, a set of 24 initial fuzzy rules was generated by heuristically identifying them from subgroup SQ definitions and practice guidelines. To improve this initial fuzzy rule set, an incremental approach was constructed to identify potential new rules from issue summaries mined from four open-source projects. The rule performance was used to determine whether the existing rule set should be updated. As a result, we obtained a final set of 99 rules[2]. To evaluate the generalizability of the obtained rule set, we evaluated it on projects that were not used in generating the rules. All metrics (accuracy, precision, recall, and f-measure) had an average above 0.8, indicating that the rule set is able to perform well in classifying issue summaries with all of the subgroup SQs. Thus, with such an automated classifier that can identify maintainability concerns expressed in issue summaries, we conduct a large empirical study to investigate the trends of maintainability across 61 open-source software projects and over 200,000 issue summaries.

3.2 Research Questions

To explore the characteristics of maintainability, we look to answer the following research questions:

- RQ1: How are software maintainability concerns expressed in different domains and ecosystems? For this RQ, each project is classified as one of the following:
  - Applications: these projects are designed to have some sort of direct interaction with general users [39]. Examples of these projects include web browsers, email clients, and office suites.
  - Infrastructure: these projects are not designed to interact with users directly. Instead, they provide facilities and services for other software to run [39]. Examples of these projects include build tools, web servers, and libraries.

  To answer this RQ, we analyze the differences for subgroup SQs between these domains as well as between Apache and Mozilla projects.

- RQ2: How are software maintainability concerns expressed across different types of issues? For this empirical study, we report on the following characteristics:

[1] Dataset can be found: https://bit.ly/2WhtWJx

[2] The final rule set can be found: http://bit.ly/3az2CRy

- Won't Fix: Issues whose resolutions are WONTFIX have been classified such that they are not planned to be fixed. This can be for a variety of reasons, such as when the issue involves an unsupported method or tool, or when the issue is not worth the cost [40].
- Reopened: Issues that have been previously closed can be reopened in cases such as when new reproducibility information is reported, previous root causes are identified as misunderstood, reports with insufficient information are updated, or the priority of the issue has been increased [41].
- Unresolved: Some issues are left unresolved without updates. To investigate these unresolved issues, we filter the lists to identify issues whose status is not resolved or closed, and whose last changed or updated date is more than a year from June 30, 2020. This date is based on the point up to when the issues were collected.
- Severity: Issues are often classified depending on their impact, with most of the projects defining them as blocker, critical, major, etc. The MozillaWiki defines severity in terms of levels: S1, S2, S3, S4 for catastrophic, serious, normal, and small/trivial respectively; however, the previously mentioned descriptors are used more commonly. Thus for this RQ, the severities are defined as follows:
  - Blocker: Blocker, S1
  - Critical: Critical
  - Major: Major, S2
  - Others: All other categories

- RQ3: How does software maintainability change as software evolves?

  To gain a better understanding of how maintainability changes as software evolves, we look to the issues of Apache Tomcat. This project has been selected as it has a long history: the Apache Bugzilla contains issues from Tomcat 3 to Tomcat 9, and it has versions separated to the patch level of granularity. To answer this RQ, we look at the data in three ways: by major version, within major version by year, and by year overall. The last updated date in the Apache Tomcat Archive is used to map each patch to a year. Evaluation by changes in minor version is not done as many versions have at most 2 minor versions for a given major version. In cases where versions have patch variants, such as release candidates or betas, the versions are combined. For example, Tomcat 4 has issues of version 4.0 Beta 1, Beta 2, etc. These are combined with the release candidates, milestones, and final version issues into a single 4.0.0 category.

3.3 Study Design

3.3.1 Study Subjects

This empirical study focuses on projects found within the Mozilla and Apache ecosystems. Table 1 provides the characteristics of the projects chosen for this study. Some projects are filtered out of the study subjects. From Mozilla, projects from other and graveyard are excluded from the study as they contain many projects that do not focus on software. From Apache, projects that contain fewer than 100 issues are excluded. Apache OpenOffice and Apache SpamAssassin have their own Bugzilla repositories which are included with the other Apache projects.

3.3.2 Data Extraction and Analysis

Issue summaries from the selected projects are downloaded from their respective Bugzilla repositories along with the issue characteristics such as version, Open Date, etc. The issues are then classified as described in Section 3.1.2. They are then separated according to the criteria described in the RQs. Issues that are identified as invalid or duplicates are filtered out to avoid over-counting. For each SQ, the overall proportion is calculated from the number of expressing issues over the total number of issues to correct for differences in the number of issues reported between groups. Relative proportion is calculated from the number of expressing issues over the total number of issues that express any maintainability concern to compare how much each subgroup SQ contributes to overall maintainability.

In total, 229,329 issues are analyzed and classified as relating to one of the maintainability subgroup SQs or as non-maintainability. Of these, 82,577 (36%) are maintainability related and 146,752 (64%) are non-maintainability related. Figure 2 shows the relative and overall proportions of each of the maintainability subgroup SQs. The most prevalent maintainability subgroup SQs are understandability, portability, and accessibility.

4 Results and Discussion

4.1 RQ1

Of the 229,329 issues analyzed and classified, 180,706 come from Mozilla systems and 48,623 from Apache systems.

Table 2 complies the number of issues expressing each SQ, the overall proportion of each SQ over the total number of issues, and the relative proportion of each SQ over the total number of maintainability issues.

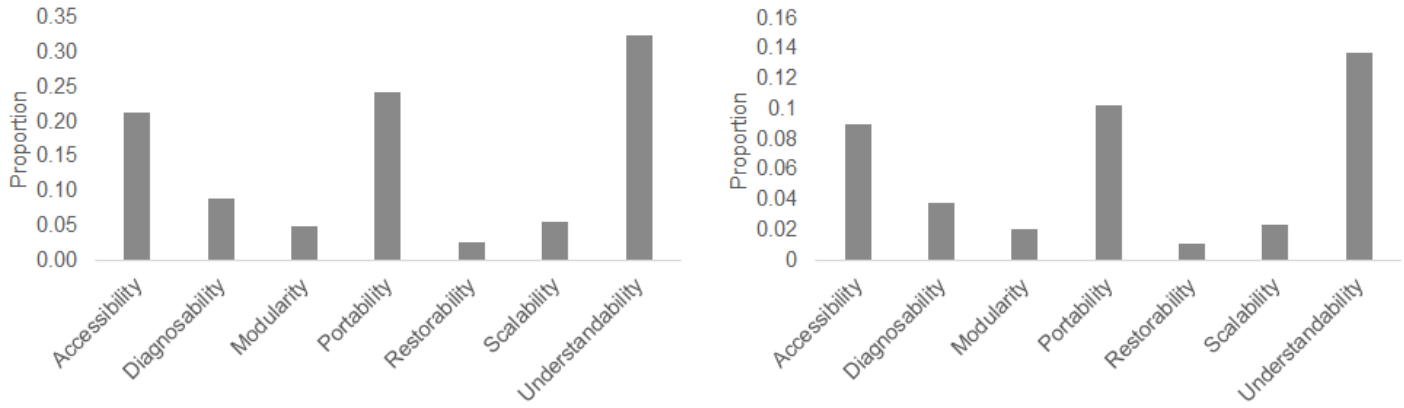
Figures 3a and 3b show the proportion of issue summaries that express maintainability concerns across the chosen ecosystems and domains respectively. There is not a large difference between the proportion of maintainability issues when considering ecosystem or domain.

4.1.1 Overall Proportions

A MANOVA is performed to examine whether there is a significant association between the overall proportion of maintainability and each maintainability subgroup SQ with ecosystems or domains. More specifically, whether ecosystems or domains have a significant effect on the overall proportions of maintainability and each maintainability subgroup SQ. While there is a significant association found between ecosystem and the overall proportions of maintainability and its subgroup SQs, $F = 19.136$, $df = 8$, $p=0.014$ (<0.05), univariate analysis does not identify statistically significant SQs that contribute to the differences between ecosystems. In addition,

Table 1: Study subject characteristics

Apache			Mozilla		
Project	Number of Issues	Domain	Project	Number of Issues	Domain
Ant	6144	Infrastructure	Bugzilla	10000	Infrastructure
Apache httpd-1.3	898	Infrastructure	bugzilla.mozilla.org	10000	Application
Apache httpd-2	8288	Infrastructure	Calendar	10000	Application
APR	818	Infrastructure	Chat Core	1295	Infrastructure
Batik	1029	Infrastructure	Cloud Services	9932	Application
BCEL	168	Infrastructure	Conduit	1713	Infrastructure
Fop	2170	Application	Core	10000	Infrastructure
JMeter	4609	Infrastructure	Data Platform and Tools	4251	Application
Lenya	1449	Application	DevTools	10000	Infrastructure
Log4j	1387	Infrastructure	Directory	726	Infrastructure
OpenOffice	10000	Application	External Software Affecting Firefox	1567	Application
POI	4667	Infrastructure	Firefox	10000	Application
Regex	102	Infrastructure	Firefox Build System	10000	Infrastructure
Security	252	Infrastructure	Firefox for Android	10000	Application
Slide	432	Application	Firefox for iOS	7067	Application
Spamassassin	7713	Application	Gecko View	2523	Infrastructure
Taglibs	764	Infrastructure	Instantbird	1709	Application
Tomcat Connectors	804	Infrastructure	JSS	455	Infrastructure
Tomcat Modules	187	Infrastructure	MailNews Core	10000	Infrastructure
Tomcat Native	178	Infrastructure	Mozilla Localizations	10000	Application
Tomcat 3	1129	Infrastructure	NSS	10000	Infrastructure
Tomcat 4	3374	Infrastructure	Remote Protocol	650	Infrastructure
Tomcat 5	3118	Infrastructure	SeaMonkey	10000	Application
Tomcat 6	1386	Infrastructure	Socorro	8822	Infrastructure
Tomcat 7	1643	Infrastructure	Testing	10000	Infrastructure
Tomcat 8	1213	Infrastructure	Testopia	920	Infrastructure
Tomcat 9	486	Infrastructure	Thunderbird	10000	Application
XercesJ	426	Infrastructure	Toolkit	10000	Infrastructure
Xindice	163	Infrastructure	Tree Management	6673	Infrastructure
			Web Compatibility	4603	Application
			WebExtensions	8344	Infrastructure
			WebTools	5682	Infrastructure



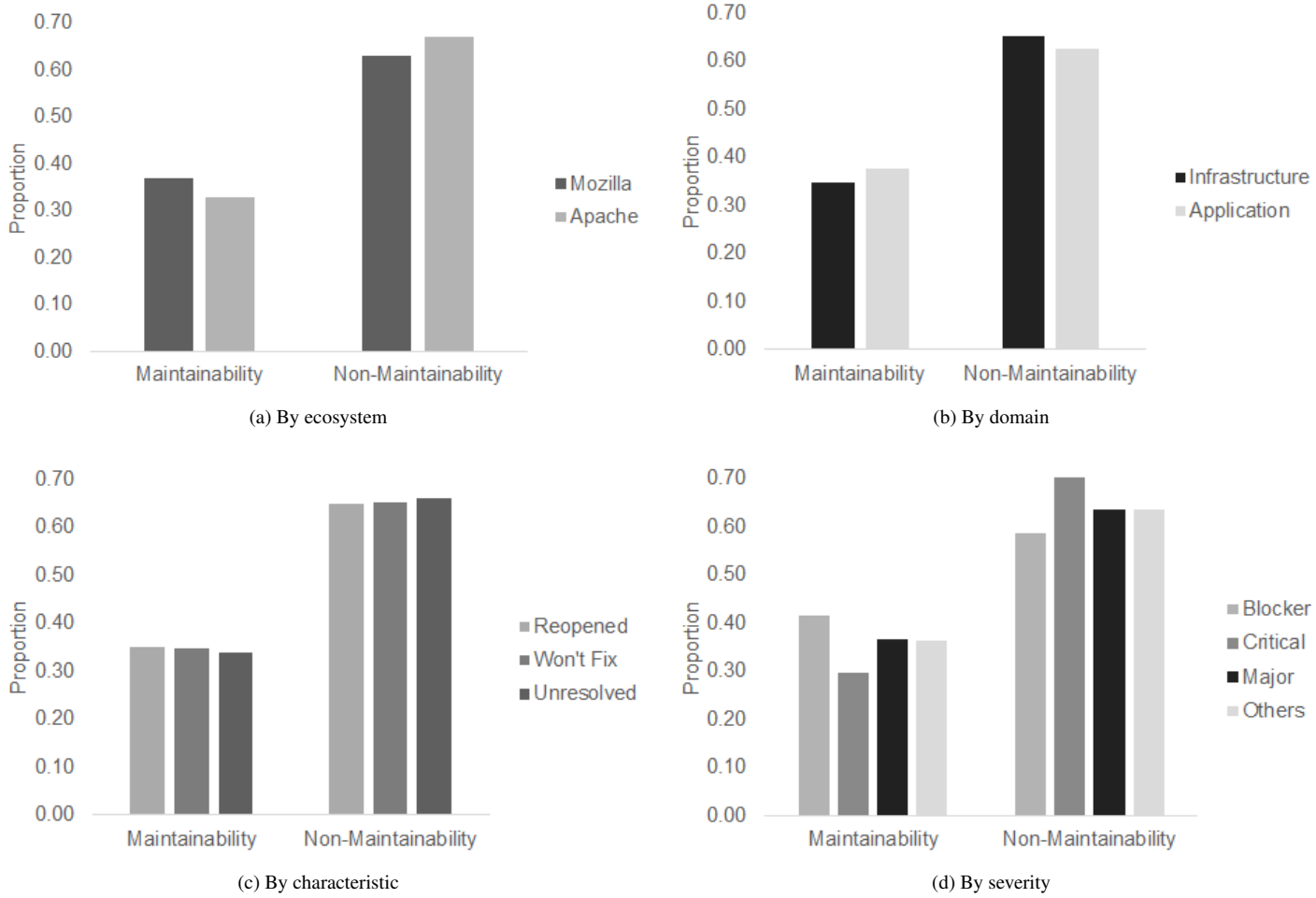
(a) Relative proportion

(b) Overall proportion

Figure 2: Proportions of subgroup SQ concerns across all issues

Table 2: SQ issue counts (#), overall proportions (O), and relative proportions (R) by various types and classifications

Type		Acc.			Dia.			Mod.			Por.			Res.			Sca.			Und.		
		#	O. %	R. %	#	O. %	R. %	#	O. %	R. %	#	O. %	R. %	#	O. %	R. %	#	O. %	R. %	#	O. %	R. %
Ecosystem	*Apache*	4506	9	25	1264	3	7	669	1	4	4021	8	22	434	1	2	1292	3	7	6143	13	34
	Mozilla	16191	9	21	7361	4	9	4079	2	5	19482	11	25	2088	1	3	4057	2	5	25201	14	32
Domain	*Application*	9586	10	22	2132	2	5	1951	2	4	12553	13	29	1116	1	3	2090	2	5	14258	14	33
	Infrastructure	11111	8	21	6493	5	12	2797	2	5	10950	8	21	1406	1	3	3259	2	6	17086	13	32
Severity	*Blocker*	572	12	24	112	2	5	117	2	5	910	19	38	43	1	2	128	3	5	507	10	21
	Critical	1550	9	24	189	1	3	369	2	6	2567	14	40	139	1	2	1072	6	17	556	3	9
	Major	2152	14	32	322	2	5	339	2	5	1936	13	29	221	1	3	581	4	9	1122	7	17
	Others	16423	9	20	8002	4	10	3923	2	5	18090	9	22	2119	1	3	3568	2	4	29159	15	36
Other Resolutions	*Reopened*	114	9	23	107	9	21	23	2	5	88	7	18	19	2	4	24	2	5	127	10	25
	Won't-Fix	1392	8	20	484	3	7	348	2	5	2017	12	29	185	1	3	305	2	4	2123	13	31
	Unresolved	1623	9	22	941	5	13	381	2	5	1450	8	20	253	1	3	492	3	7	2295	12	31



(a) By ecosystem

(b) By domain

(c) By characteristic

(d) By severity

Figure 3: Distributions of maintainability versus non-maintainability issues

no statistically significant results are found for the overall proportions of maintainability and its subgroup SQs in terms of different domains, *F = 13.036, df=8, p=0.192 (>0.05).*

4.1.2 Relative Proportions

Pearson's Chi-squared tests are performed to examine whether the ecosystems or the domains are associated with the distributions of the relative proportion of each SQ. Overall, the distributions of the relative proportion of each SQ differ significantly in domains, χ^2 (6, N = 61) = 2226.5, p < 0.001; and also in ecosystems, χ^2 (6, N = 61) = 2921.2, p < 0.001.

As shown in Table 2, understandability, portability, and accessibility are most prevalent expressed concerns. When considering ecosystem, these SQs comprise the majority for both Mozilla and Apache; however, the Apache systems tend to express more accessibility concerns than portability concerns. In considering domain, the same three SQs are the most prevalent; however, for infrastructure type software, there tend to be more diagnosability issues and fewer relating to portability when compared to application type software.

Summary of RQ1: To summarize, in this study, there is a statistically significant association between ecosystem and the overall proportions of maintainability and its subgroup SQs. The distributions of relative proportions of subgroup SQs differ significantly between ecosystems and domains; there is a trend in application software which tends to have more portability issues and fewer diagnosability issues compared to infrastructure software. As application type software is targeted toward end-users, there may be a larger variety of use cases which would necessitate compatibility with other software. In contrast, infrastructure software is targeted toward developers. In this case, they may place more importance on being able to diagnose issues with the software and may already be aware of incompatibilities with other software.

4.2 RQ2

4.2.1 RQ2a: Reopened, Won't-fix and Unresolved

Of the 229,329 issues classified, 1,251 are marked as REOPENED, 16,909 are marked as WONTFIX, and 18,809 are identified as unresolved. Figure 3c shows the proportion of issue summaries that expressed maintainability concerns across these three categories. Similar to the overall proportion, these categories are comprised of about 35% maintainability issues.

Table 2 compiles the number of issues expressing each SQ, the overall proportion that these SQs make of all tagged issues, and

the relative proportion that these SQs make of all maintainability related issues across categories. For won't-fix and unresolved issues, understandability, portability, and accessibility make up the largest percentage of expressed maintainability concerns. For reopened issues, diagnosability replaces portability of the top 3 subgroup SQ concerns. Won't-fix issues tend to express more portability concerns with relatively fewer diagnosabilty concerns compared to the other two categories.

4.2.2 RQ2b: Severity

Comparing the number of issues within each category of severity, the issues are divided into 4887, 17799, 15424, and 191219 issues associated with Blocker, Critical, Major, and Others types respectively. Figure 3d shows the proportion of issue summaries expressing maintainability concerns across these different levels of severity. For the highest severity category, blocker, maintainability issues make up 41%. Interestingly, the relative proportion of maintainabilty issues decreases for the next severity category, Critical, to 30%, while increasing again to 36% in the Major and Others categories. Table 2 compiles the number of issues expressing each SQ, the overall proportion that these SQs make of all tagged issues, and the relative proportion that these SQs make of all maintainability related issues across severities.

Portability, accessibility, and understandability remain the most prevalent expressed SQs except for the Critical category, where understandability is replaced by scalability. Blocker and Critical issues have similar proportions of portability and accessibility issues while accessibity issues have higher prevalence in Major issues. Finally, the less severe Others category is comprised largely of understandability issues.

A MANOVA is performed to examine whether there is a significant association between the overall proportion of maintainability and each maintainability subgroup SQs with different levels of severity. There is a statistically significant association found between levels of severity and the overall proportions of maintainability and its subgroup SQs, $F = 166.42$, $df = 24$, $p < 0.001$. Of the subgroup SQs, accessibility, portability, scalability, and understandability are found to have statistically significant differences across severity levels, with $p < 0.001$ after Bonferroni correction.

Pearson's Chi-squared tests are performed to examine whether levels of severity are associated with the distributions of the relative proportion of each SQ. Overall, the distributions of the relative proportion of each SQ differ significantly across severity, χ^2 (18, N = 61) = 5795.8, $p < 0.001$.

Summary of RQ2: To summarize, in this study, won't-fix and unresolved issues tend to express understandability, portability, and accessibility concerns. Reopened issues tend to express diagnosability concerns in addition to understandability and accessibility. For the case of won't-fix issues tending to express more portability concerns than the baseline-total, this result could be explained as portability issues involve factors external to the system. These types of issues are more likely to involve unsupported tools or potentially costly integrations, leading to a classification of WONTFIX.

The highest severity issues tend to have a higher proportion of maintainability issues than lower severity issues, and there is a significant association between levels of severity and the overall and relative proportions of the different subgroup SQs. This finding validates our results from the previous empirical study. As these issues have a high impact on the system, this reinforces the importance of ensuring high maintainability to avoid these types of issues.

4.3 RQ3

4.3.1 Changes between major versions

When comparing between major versions, there does not appear to be a strong trend in terms of the percentage maintainability issues make of the total. Figure 4a shows the relative proportions of each subgroup SQ. There appears to be a decreasing trend for accessibility and an increasing trend for understandability related issues for later versions. Figure 4b shows the overall proportions of each subgroup SQ. In this case, the increase in understandability related issues continues.

4.3.2 Changes within major version by year

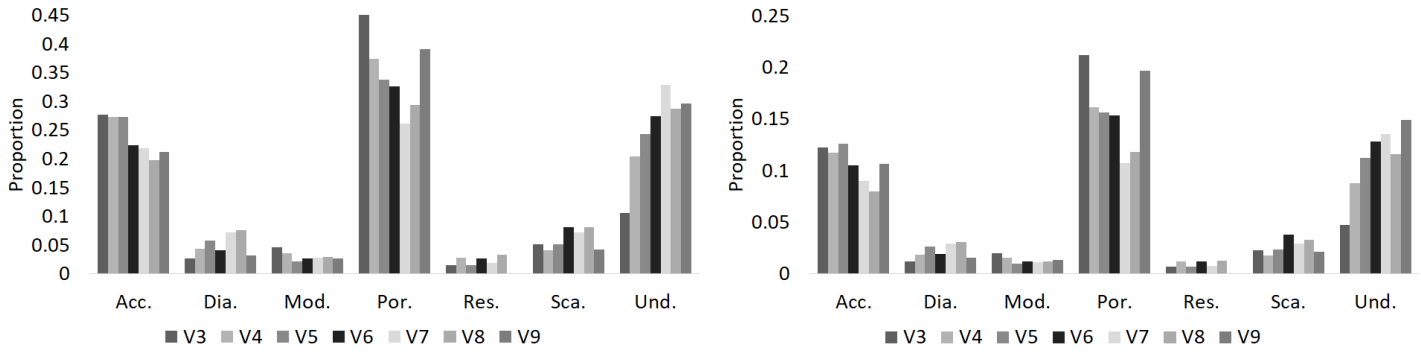
To provide an analysis of the relationship between the subgroup SQs and time, we map each patch to a year based on the last updated date in the Apache Tomcat Archive and perform linear regression within each major version. Figures 5a, and 5b show the statistically significant trends ($p < 0.05$) in relative proportions, overall proportions, and number of issues reported respectively. For clarity, the scales of the y-axes are set individually by version in Figure 6 due to relatively large differences in overall proportions and number of issues between versions.

- Version 3: Only restorability showed a statistically significant decline in number of issues reported. Relative and overall proportions did not have statistically significant relationships. We acknowledge that very few issues overall were reported in 2003 and 2004 which may contribute to this trend.

- Version 4: Other than modularity, all subgroup SQs showed statistically significant decline in number of issues reported. However, the decline in number of issues reported is also present for issues in general. In terms of relative proportions, accessibility and understandability showed declines while portability showed an increase over time. For overall proportions, accessibility, modularity, restorability, and understandability all showed declines. We acknowledge that very few issues overall were reported from 2005 to 2008 which results in the relative proportions of 0 for accessibility and understandability and the relative proportions of 1 in portability within those years. These values may skew the significance of the trends.

- Version 5: Portability and understandability showed statistically significant decline over years for number of issues reported. Those SQs showed similar declines, and accessibility showed statistically significant increase for relative and overall proportion of SQs.

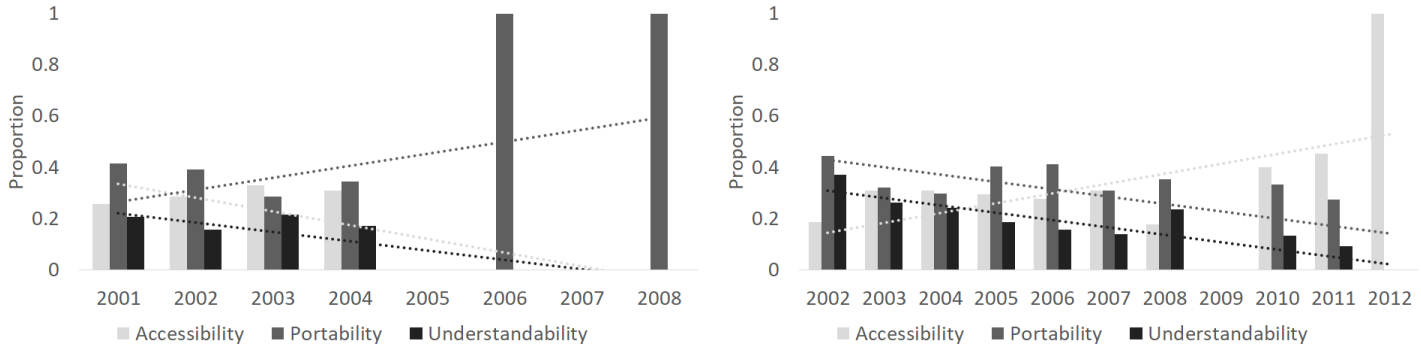
- Version 6: Accessibility and portability showed a statistically significant decrease in number of issues reported over years.



(a) Relative proportion

(b) Overall proportion

Figure 4: Proportions by major version of Apache Tomcat



(a) Version 4

(b) Version 5

Figure 5: Statistically significant relative proportions of subgroup SQ concerns across years by versions

This was shared with the number of maintainability, non-maintainability, and total issues reported. No SQs were found to have a statistically significant change in relative proportion; however, the overall proportion of portability related issues increased.

- Version 7: Accessibility, portability, scalability, and understandability all had declines over time along with decreases in numbers of maintainability, non-maintainability, and total issues reported. Overall proportions of portability and maintainability had increases over time, but no trends were found for relative proportions.
- Version 8: Accessibility, portability, and understandability all had declines over time along with decreases in number of maintainability, non-maintainability, and total issues reported. No trends were found for relative or overall proportions.
- Version 9: No significant trends were found for version 9

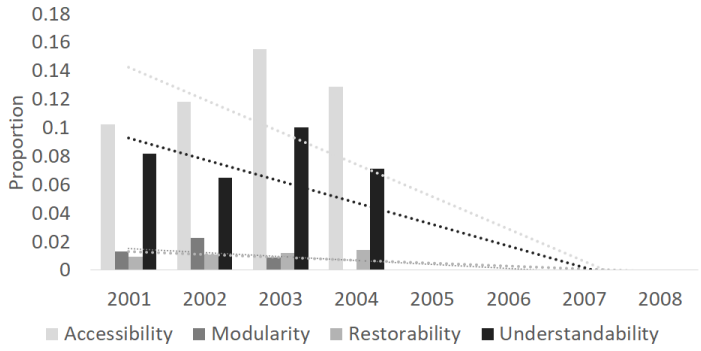
4.3.3 Changes by year overall

When considering all issues by year, there is a general decline in the number of issues reported overall which is found for all SQs. This trend is found in general for the number of issues reported; Figure 6 shows the number of maintainability and non-maintainability related issue summaries reported per year for Tomcat. However, when looking at the overall and relative proportions, there are no statistically significant trends across the SQs.

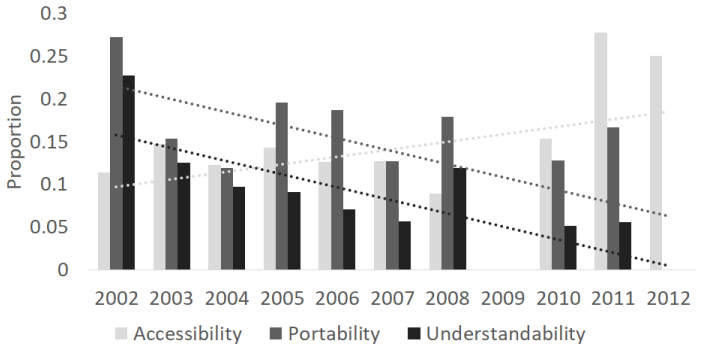
Summary of RQ3: To summarize, although there is not a significant trend in terms of the percentage of maintainability issues between major versions, there is a decreasing trend for accessibility while an increasing trend for understandability for later versions. In addition, various subgroup SQs show statistically significant trends in relative proportions, overall proportions and the number of issues reported.

While there are a number of statistically significant trends within versions, the most common is a decline in number of issues reported overall over time and within the versions themselves. Possible reasons for the decline in issues reported include that people are not reporting as many issues in general compared to in the past, the later versions are still being maintained and developed so there has been less time to report issues, and finally that the maintainability has increased over time.

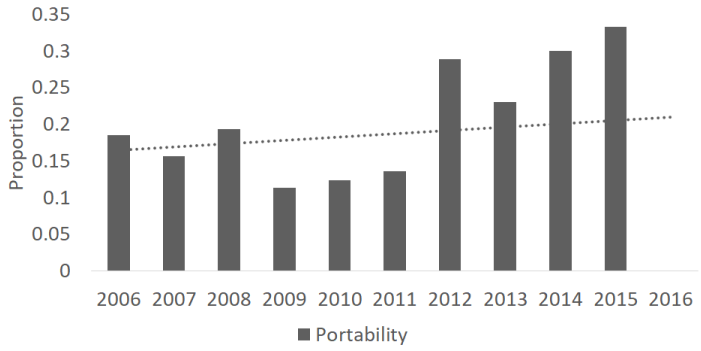
As there are no statistically significant trends in terms of relative or overall proportions of the subgroup SQs when looking at changes by year overall, this may indicate that focusing on the greatest relative proportions overall (i.e. accessibility, portability, and understandability) will be effective regardless of time in the life cycle.



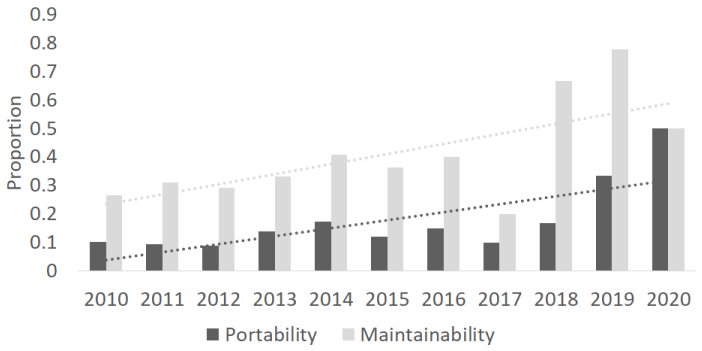
(a) Version 4



(b) Version 5



(c) Version 6



(d) Version 7

Figure 6: Statistically significant overall proportions of subgroup SQ concerns across years by version

4.4 Threats to Validity

This study depends on the model developed in [38]. The accuracy of classification is subject to the limitations and threats to validity detailed in the prior work.

Some information in issue summaries is self-reported by the developers of the different software projects such as severity, version information, etc. Validation of this information is out of the scope of this study; however, the developers reporting the issues are the most qualified to assess these metrics given their familiarity with the projects. Thus, we assume the reported information is correctly identified.

As our case study focused only on Apache Tomcat, our findings with regard to quality changes within versions and over the life of the project should not be generalized to other projects without further study.

5 Conclusion

Motivated by the lack of effective systematic measurement of maintainability in practice, we presented a novel approach to achieve automatic identification on how software maintainability and its subgroup SQs are expressed in a series of publications. Enabled by the automated approach to scale up analysis of maintainability through issue summaries, in this article, a large empirical study on 229,329 issue summaries from 61 different projects was conducted. Out of all the issue summaries, 82,577 issues were classified as expressing maintainability concerns. These issues were further analyzed to evaluate the differences between domains, ecosystems, and types. We found differences in relative proportions across ecosystems, domain and issue severity. Additional analysis was performed on Apache Tomcat to evaluate the evolution of maintainability across several versions. We identified several trends within versions and over time, such as a general decline in the number of issues reported overall in all the subgroup SQs and a statistically significant decline in portability and accessibility in multiple versions.

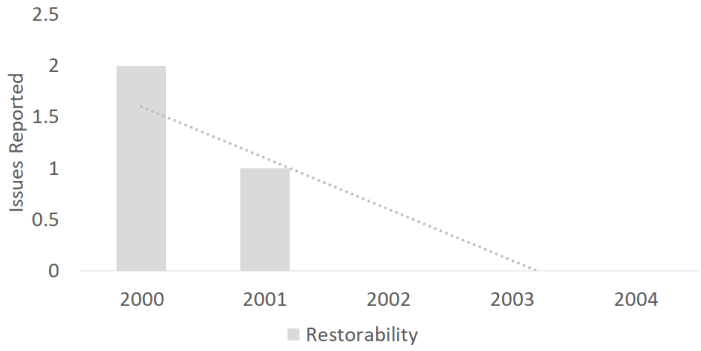
We believe that our work introduces a new angle to the area of software maintainability evaluation, encourages researchers to utilize unstructured software artifacts, and promotes automated solutions to incorporate standards and frameworks into software development process.

Conflict of Interest The authors declare no conflict of interest.

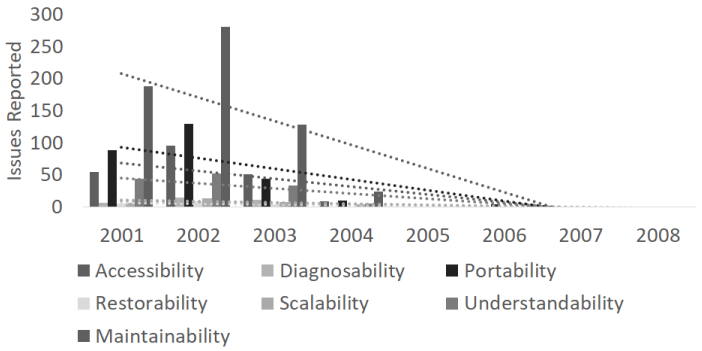
Acknowledgment This material is based upon work supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract HQ0034-13-D-0004. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

References

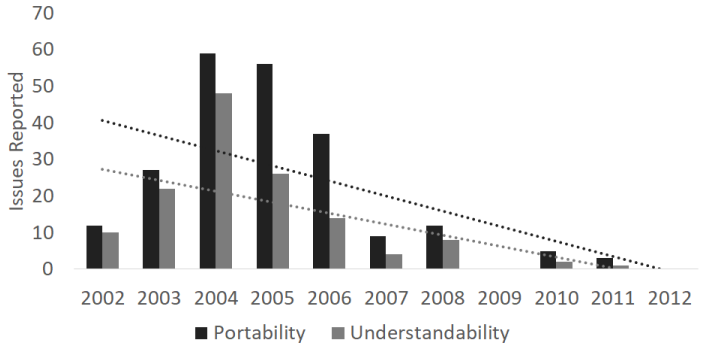
[1] J. Koskinen, "Software maintenance fundamentals," Encyclopedia of Software Engineering, P. Laplante, Ed., Taylor & Francis Group, 2009.



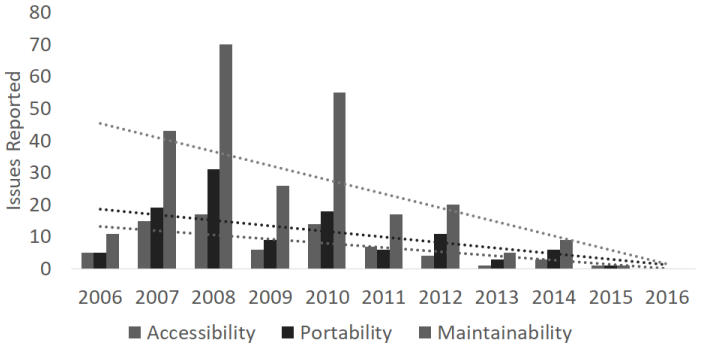
(a) Version 3



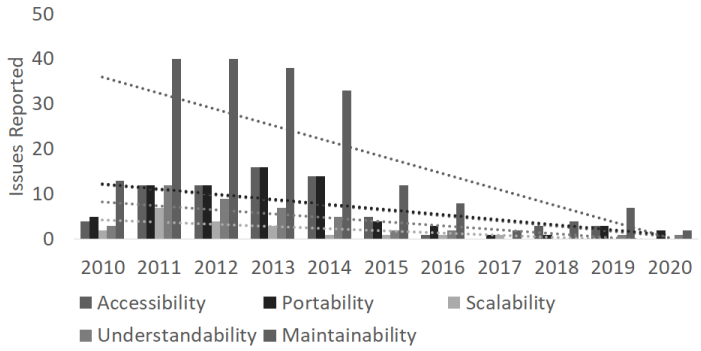
(b) Version 4



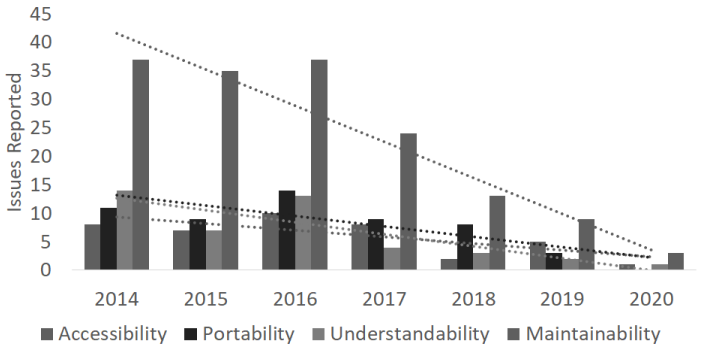
(c) Version 5



(d) Version 6



(e) Version 7



(f) Version 8

Figure 7: Statistically significant number of issues per subgroup SQ concerns across years by version

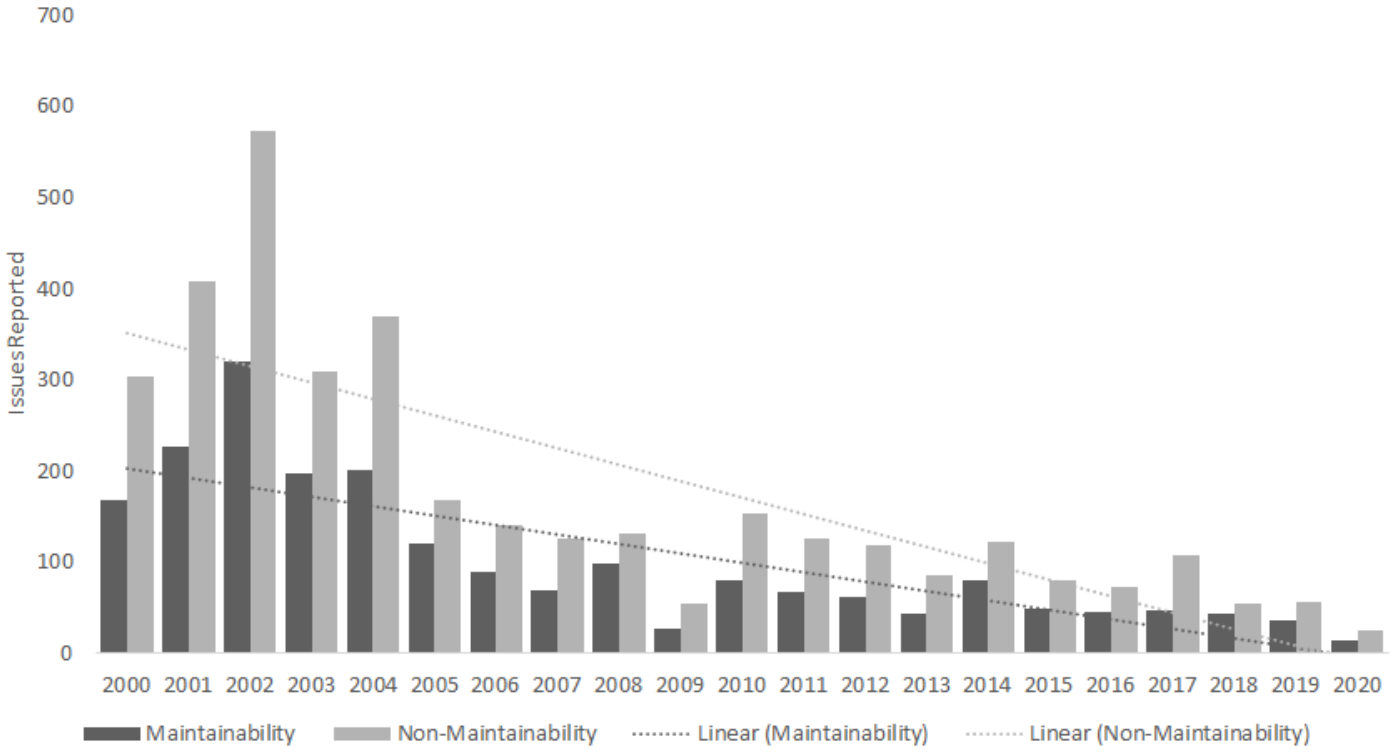


Figure 8: Number of issues reported by year

[2] B. Boehm, C. Chen, K. Srisopha, L. Shi, "The key roles of maintainability in an ontology for system qualities," in INCOSE International Symposium, **26**, 2026–2040, Wiley Online Library, 2016.

[3] M. Y. Shoga, C. Chen, B. Boehm, "Recent Trends in Software Quality Interrelationships: A Systematic Mapping Study," in 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 264–271, 2020, doi:10.1109/QRS-C51114.2020.00052.

[4] S. R. Chidamber, D. P. Darcy, C. F. Kemerer, "Managerial use of metrics for object-oriented software: An exploratory analysis," IEEE Transactions on software Engineering, **24**(8), 629–639, 1998.

[5] R. Mo, Y. Cai, R. Kazman, L. Xiao, Q. Feng, "Decoupling level: a new metric for architectural maintenance complexity," in Proceedings of the 38th International Conference on Software Engineering, 499–510, ACM, 2016.

[6] H. Wu, L. Shi, C. Chen, Q. Wang, B. Boehm, "Maintenance effort estimation for open source software: A systematic literature review," in Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on, 32–43, IEEE, 2016.

[7] L. Yu, A. Mishra, "An empirical study of Lehman's law on software quality evolution," 2013.

[8] C. Chen, R. Alfayez, K. Srisopha, L. Shi, B. Boehm, "Evaluating Human-Assessed Software Maintainability Metrics," in Software Engineering and Methodology for Emerging Domains, 120–132, Springer, 2016.

[9] B. W. Boehm, R. Madachy, B. Steece, et al., Software cost estimation with Cocomo II with Cdrom, Prentice Hall PTR, 2000.

[10] M. Riaz, E. Mendes, E. Tempero, "A systematic review of software maintainability prediction and metrics," in Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 367–377, IEEE Computer Society, 2009.

[11] S. Scalabrino, G. Bavota, C. Vendome, M. Linaresvasquez, D. Poshyvanyk, R. Oliveto, "Automatically assessing code understandability: How far are we?" in Ieee/acm International Conference on Automated Software Engineering, 417–427, 2017.

[12] C. Chen, S. Lin, M. Shoga, Q. Wang, B. Boehm, "How do defects hurt qualities? an empirical study on characterizing a software maintainability ontology in open source software," in 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), 226–237, IEEE, 2018.

[13] P. Oman, J. Hagemeister, "Metrics for assessing a software system's maintainability," in Software Maintenance, 1992. Proceedings., Conference on, 337–344, IEEE, 1992.

[14] K. D. Welker, "The software maintainability index revisited," CrossTalk, **14**, 18–21, 2001.

[15] E. VanDoren, "Maintainability Index Technique for Measuring Program Maintainability. Software Engineering Institute," 2002.

[16] D. I. Sjøberg, B. Anda, A. Mockus, "Questioning software maintenance metrics: a comparative case study," in Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, 107–110, ACM, 2012.

[17] R. Baggen, J. P. Correia, K. Schill, J. Visser, "Standardized code quality benchmarking for improving software maintainability," Software Quality Journal, **20**(2), 287–307, 2012.

[18] L. Kumar, S. K. Rath, A. Sureka, "Using Source Code Metrics and Multivariate Adaptive Regression Splines to Predict Maintainability of Service Oriented Software," in 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), 88–95, 2017, doi:10.1109/HASE.2017.11.

[19] R. Malhotra, A. Chug, "Software maintainability prediction using machine learning algorithms," Software Engineering: An International Journal (SEIJ), **2**(2), 2012.

[20] S. Jha, R. Kumar, L. Hoang Son, M. Abdel-Basset, I. Priyadarshini, R. Sharma, H. Viet Long, "Deep Learning Approach for Software Maintainability Metrics Prediction," IEEE Access, **7**, 61840–61855, 2019.

[21] M. Schnappinger, M. H. Osman, A. Pretschner, A. Fietzke, "Learning a Classifier for Prediction of Maintainability Based on Static Analysis Tools," in 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 243–248, 2019, doi:10.1109/ICPC.2019.00043.

[22] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, C. Zhai, "Have things changed now?:an empirical study of bug characteristics in modern open source software," in The Workshop on Architectural and System Support for Improving Software Dependability, 25–33, 2006.

[23] Y. Zhou, Y. Tong, R. Gu, H. Gall, "Combining text mining and data mining for bug report classification," Journal of Software: Evolution and Process, **28**(3), 150–176, 2016, doi:10.1002/smr.1770.

[24] "Detecting Missing Information in Bug Descriptions," ESEC/FSE 2017, 396–407, New York, NY, USA, 2017, doi:10.1145/3106237.3106285, event-place: Paderborn, Germany.

[25] D. C. Das, M. R. Rahman, "Security and Performance Bug Reports Identification with Class-Imbalance Sampling and Feature Selection," in 2018 Joint 7th International Conference on Informatics, Electronics Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision Pattern Recognition (icIVPR), 316–321, 2018.

[26] B. J. Guarraci, "Instrumenting software for enhanced diagnosability," 2012, uS Patent 8,141,052.

[27] D. Yuan, J. Zheng, S. Park, Y. Zhou, S. Savage, "Improving software diagnosability via log enhancement," in Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 3–14, 2011.

[28] Y. Le Traon, F. Ouabdesselam, C. Robach, "Software diagnosability," in Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on, 257–266, IEEE, 1998.

[29] A. Kavcic, "Software Accessibility: Recommendations and Guidelines," in The International Conference on Computer As A Tool, 1024–1027, 2006.

[30] J. Manual, "Manual for the operation of the joint capabilities integration and development system," US Department of Defense. Washington. DC, 2012.

[31] C. Chen, M. Shoga, B. Li, B. Boehm, "Assessing Software Understandability in Systems by Leveraging Fuzzy Method and Linguistic Analysis," in 2019 Conference on Systems Engineering Research (CSER) (2019 CSER), Washington, USA, 2019.

[32] K. J. Sullivan, W. G. Griswold, Y. Cai, B. Hallen, "The structure and value of modularity in software design," in ACM SIGSOFT Software Engineering Notes, **26**, 99–108, ACM, 2001.

[33] R. Sanchez, J. T. Mahoney, "Modularity, flexibility, and knowledge manage-ment in product and organization design," Strategic Management Journal, **17**(S2), 63–76, 2015.

[34] C. U. Smith, L. G. Williams, Performance solutions: a practical guide to creating responsive, scalable software, 1, Addison-Wesley Reading, 2002.

[35] L. Duboc, D. Rosenblum, T. Wicks, "A framework for characterization and analysis of software system scalability," in Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 375–384, ACM, 2007.

[36] J. D. Mooney, "Issues in the specification and measurement of software portability," in 15th International Conference on Software Engineering, Baltimore, 1993.

[37] A. S. Tanenbaum, P. Klint, W. Bohm, "Guidelines for software portability," Software: Practice and Experience, **8**(6), 681–698, 1978.

[38] C. Chen, M. Shoga, B. Boehm, "Characterizing software maintainability in issue summaries using a fuzzy classifier," in 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), 131–138, IEEE, 2019.

[39] R. Moazeni, D. Link, C. Chen, B. Boehm, "Software Domains in Incremental Development Productivity Decline," in Proceedings of the 2014 International Conference on Software and System Process, ICSSP 2014, 75–83, Association for Computing Machinery, New York, NY, USA, 2014, doi: 10.1145/2600821.2600830.

[40] Q. Wang, "Why Is My Bug Wontfix?" in 2020 IEEE 2nd International Work-shop on Intelligent Bug Fixing (IBF), 45–54, 2020.

[41] T. Zimmermann, N. Nagappan, P. J. Guo, B. Murphy, "Characterizing and predicting which bugs get reopened," in 2012 34th International Conference on Software Engineering (ICSE), 1074–1083, 2012.