The Relation between Software Maintainability and Issue Resolution Time: A Replication Study

Wijnmaalen, J.; Chen, C.; Bijlsma, D.; Oprescu, A.-M.

Publication date
2019

Document Version
Final published version

Published in
SATTOSE 2019 : Seminar Series on Advanced Techniques & Tools for Software Evolution

License
CC BY

The Relation between Software Maintainability and Issue Resolution Time: A Replication Study

Joren Wijnmaalen
University of Amsterdam
Amsterdam, The Netherlands
j.wijnmaalen@protonmail.com

Cuiting Chen
Software Improvement Group
Amsterdam, The Netherlands
c.chen@sig.eu

Dennis Bijlsma
Software Improvement Group
Amsterdam, The Netherlands
d.bijlsma@sig.eu

Ana-Maria Oprescu
University of Amsterdam
Amsterdam, The Netherlands
a.m.oprescu@uva.nl

Abstract

Higher software maintainability comes with certain benefits. For example, software can be updated more easily to embrace new features or to fix bugs. Previous research has shown that there is a positive correlation between the maintainability score measured by the SIG maintainability model and shorter issue resolution time. This study, however, dates back to 2010. Eight years later, the software industry has evolved with a fast pace, as well as the SIG maintainability model. We would like to rerun the experiment to test if the previously found relations are still valid.

When remeasuring the maintainability of the systems with the *new* version of the SIG maintainability model (2018), we find that majority of the systems score lower maintainability ratings. The overall maintainability correlation with defect resolution time decreased significantly while the original system properties correlate similar with defect resolution time compared to the original study.

In: Anne Etien (eds.): Proceedings of the 12th Seminar on Advanced Techniques Tools for Software Evolution, Bolzano, Italy, July 8-10 2019, published at http://ceur-ws.org

1 Introduction

The definition for software quality has been standardized by the International Organization for Standardization (ISO) since 2001 in their document ISO 9126 [ISO11b]. Since then, the definition has undergone a variety of changes as it has been revised into the ISO 25010 in 2011 [ISO11a]. The standard decomposes software quality into a set of characteristics. Software maintainability is one of such characteristics.

Research has shown the importance of high software maintainability. Bakota et al. found an exponential relationship between maintainability and cost [BHL+12]. Bijlsma and Luijten showed a strong positive correlation between software maintainability and issue resolution time [BFLV12]. Maintenance activities largely involve solving issues that arise during development or when the product is in-use. A better maintainable code base decreases the amount of time needed to resolve such issues.

However, the study by Bijlsma and Luijten dates back to 2012. To assess the maintainability of systems, they made use of the maintainability model developed by the Software Improvement Group (SIG), dating back to 2010. This model refers to ISO 9126 for their definition of software quality, more specifically, maintainability. Over the years the SIG maintainability model has been evolving (a new model has been announced in 2018 [sig]), implementing a variety of smaller changes along the new software quality definition as documented in ISO 25010. Furthermore, the software industry has been evolving at a fast pace. The oldest systems Bijlsma and Luijten assessed for their empirical results date back to the beginning of

the 2000s. A lot has changed in the software industry since then. Both the landscape of technologies has changed, as well as the processes around software. For example, DevOps has emerged since the mid 2010s, introducing concepts such as continuous integration and delivery. These concepts, potentially, change the way how issues are being resolved as integration is being largely automated instead of being a manual action.

Around the broader question: *"What is the relation between software maintaina bility and issue resolution time?"*, we propose the following research question:

- **RQ1.1** Does the previously found strong correlation between maintainability and issue resolution time still hold given the latest (2018) SIG maintainability model?

2 Background

2.1 The SIG Maintainability Model

The ISO 25010 standard defines Software Quality through a range of quality characteristics. Each of these characteristics is further subdivided into a set of sub-characteristics. Software maintainability is one of such characteristics and is further subdivided into the following sub-characteristics: *analyzability, modifiability, testability, modularity and reusability.* The standard, however, does not provide how to directly measure the various quality characteristics and sub-characteristics. Instead, the Software Improvement Group (SIG) provides a pragmatic model to directly assess maintainability through the static analysis of source code [HKV07]. The SIG maintainability model lists a set of source code metrics, also called software product properties. The following software product properties are measured: *volume, duplication, unit size, unit complexity, unit interfacing, module coupling, component balance* and *component independence.* These product properties are then mapped to the sub-characteristics as defined in the ISO 25010 standard. These mappings, what product properties influence what characteristics, are based on expert opinion. Table 1 illustrates these mappings.

To calculate the maintainability rating of a system, the model first measures the product properties. These raw measures are converted to a star based rating based on a benchmark internal to SIG (1 to 5 stars, where 3 stars is the market average). Note, the stars do not divide the distribution of systems into even buckets, instead 5% of systems are assigned one star, 30% two, 30% three, 30% four and 5% five stars. Secondly, the product property ratings are aggregated into the maintainability sub-characteristic ratings based on the relations as defined in Table 1. Finally, the sub-characteristics ratings are all aggregated into a single final maintainability rating.

Table 1: Relationship between software product properties and the ISO 25010 maintainability sub-characteristics. Data taken from SIG/TViT Evaluation Criteria Trusted Product Maintainability [Vis18]

	Volume	Duplication	Unit Size	Unit Complexity	Unit Interfacing	Module Coupling	Component Balance	Component Independence
Analyzability	X	X	X				X	
Modifiability		X		X		X		
Testability	X			X				X
Modularity						X	X	X
Reusability			X		X			

Evolution of the SIG Maintainability Model

Both Bijlsma and Luijten assessed the maintainability characteristic of software quality as described by the ISO 9126 standard using the SIG maintainability model. Since the ISO 9126 standard has been revised into the ISO 25010 standard, the SIG maintainability model has evolved accordingly, as is part of the motivation for this replication study. In order to reason about the results of this replication study, the differences between the 'modern' SIG maintainability model (hereinafter referred to as the new model) and the model used by Bijlsma and Luijten (hereinafter referred to as the old model) need to be highlighted.

Compared to ISO 9126, ISO 25010 adds the sub-characteristic *modularity* to maintainability. Modularity is defined as "The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components." [ISO11a]. In order to account for this new sub-characteristic, two new system properties were introduced in the new model: *component balance* and *component independence.* Apart from accounting for the new sub-characteristic, these properties were expected to stimulate discussions about the architecture of systems and to incorporate a common viewpoint in the assessment of implemented architectures, as mentioned by Bouwers et al. in their evaluation of the SIG maintainability model metrics [BvDV13].

Introduction of these properties also raises questions on the definition for a component. Visser defines

the term component as the following in his technical report: "A component is a subdivision of a system in which source code modules are grouped together based on a common trait. Often components consist of modules grouped together based on a shared technical or functional aspect" [Vis18]. In practice, this definition still deems too vague. It introduces the need for an external evaluator to point out the core components of any specific system, based on their perception on how functionality is grouped and it's granularity.

2.2 Issue Resolution Time

Both Luijten and Bijlsma look at issue resolution time in their studies. Bijlsma defines issue resolution time as "the total time an issue is in open state. [...] Resolution time is not simply the time between the issue being reported and the issue being resolved." [BFLV12] Instead, Bijlsma illustrates the life cycle of an issue using Figure 1. Bijlsma measured the highlighted period of time in the Figure for his study. Even though it would seem better to start measuring when the status of an issue is set to assigned, this was realistically not a possibility for the data Bijlsma obtained. Many projects Bijlsma analyzed were inconsistent in using the assigned property in their Issue Tracking Systems (ITS), making it impossible to accurately determine when a developer started working on an issue.

Next to the issue resolution time life cycle, there is also the notion of issue types. Various issue tracking systems use different terms to denote the variety in issues. Bijlsma defined the following types: *defect, enhancement, patch and task*. A defect, according to Bijlsma, is a "problem in the system" [BFLV12]. An enhancement can be "the addition of a new feature, or an improvement of an existing feature". Tasks and patches are "usually one time activities" and unify various other issue types with a range of urgencies. The tools Bijlsma and Luijten used in their experiment normalized all issues obtained from the various ITS's towards these four types only. Luijten originally focussed on issues of type defect, where Bijlsma expanded with issues of type enhancement.

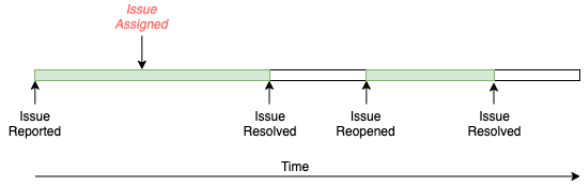

Figure 1: The issue resolution time (in green) as measured by Bijlsma and Luijten [BFLV12]

2.3 Issue Resolution Quality Ratings

Given the definition for the issue resolution time metric, both Luijten and Bijlsma collected measurements from various projects. In order to compare these resolution times on a project level, the resolution times per issue need to be aggregated. Intuitively, statistical properties such as the mean come to mind. However, as Luijten points out, the resolution times collected are not normally distributed [Lui10]. Therefore, Luijten created various risk categories, such that the issue resolution distribution is divided into buckets, choosing thresholds such that the buckets are filled equally. Table 2 illustrates the risk categories with their thresholds as defined by Luijten for issues of type defect. Similar thresholds are defined for issues of type enhancement.

Based on these categories Luijten continues to define *quality ratings*, to further align with the rating system the SIG maintainability model implements. Table 3 shows the mapping between risk categories and quality ratings. The thresholds are chosen such that 5% of the systems will receive a 5-star rating, 30% four stars, 30% three, 30% two, and 5% one star (the same distribution as the SIG maintainability model uses, Section 2.1). For measurement purposes, these star ratings are interpolated between the interval [0.5, 5.5], as is standard in the SIG maintainability model as well.

Table 2: Luijten's risk categories for issues of type defect [Lui10]. Lower risk means faster issue resolution times

Category	Threshold(days)
Low	0 - 23.6
Moderate	23.6 - 68.2
High	68.2 - 198
Very High	198+

Table 3: Luijten's rating thresholds for issues of type defect [Lui10]. Higher ratings means faster issue resolution time.

Rating	Moderate	High	Very High
*****	7%	0%	0%
****	25%	25%	2%
***	43%	39%	13%
**	43%	42%	35%

3 Method

This replication study aims to discover if the relationship between maintainability and issue resolution

times still hold using the *new* SIG maintainability model to assess maintainability. Therefore, the same systems and issue tracking data will be used as in Bijlsma's experiment. Bijlsma assessed 10 open source systems looking at multiple snapshots situated in various points of time of the systems lifespan. Given the definition for issue resolution time, and the concept of issue resolution time quality ratings, as described in Section 2.3, ratings are calculated per snapshot. For each snapshot, Bijlsma and Luijten consider all issues that are closed and/or resolved between that snapshot and the next as relevant for that snapshot [Lui10]. These ratings are directly re-used in this experiment as calculated by Bijlsma, as they were archived and directly ready for use.

Table 4 shows the systems assessed by Bijlsma. Since the original snapshots Bijlsma used in his study were not archived, the snapshots had to be re-obtained. For every snapshot Bijlsma listed a version and a date. Using this data we were able to retrieve all snapshots by using the following two methods for retrieval:

- **Official System Archives** Some systems maintain an official archive. Snapshots matching date and version number are directly retrieved from these archives. For a small amount of snapshots the date Bijlsma listed deviate a couple of days from the date coupled with the version number in the archive. These snapshots were still retrieved as it was assumed that deviation in dates was caused by human error.

- **Version Control Systems (VCS)** If a systems organization stopped hosting, or does not have an archive containing older versions, the snapshot was retrieved by traversing the system's respective VCS. The majority of the systems assessed by Bijlsma make use of Subversion as their main VCS. Subversion, by default, contains the root folders '/trunk', '/branches' and '/tags'. Where trunk is the directory where the main development takes place and branches contain the features that parallel main development, the tags directory is specifically interesting as it contains read only copies of the source code in a specified set of time.

  Note that the Subversion repositories default to the trunk, branches and tags directories but are not limited to this structure. For example, Webkit adds the '/releases' directory to the repositories root, containing all major releases (where tags are of a finer granularity). In this case, both the tags and releases are navigated to find the right snapshot.

  In a small amount of cases the matching snapshots were not listed in the tags, releases or other archiving Subversion root directories. In that case, the Subversion trunk directory was checked out at the revision closest to the date listed by Bijlsma.

For every snapshot new maintainability ratings are calculated. These ratings are obtained using the SIG software analysis toolkit (SAT). The SAT implements the latest (2018) version of the SIG maintainability model. It provides the final maintainability rating, along with its sub-characteristics as described by the ISO 25010 standard.

3.1 Data Acquisition

In order to replicate Bijlsma's original study, the same data is needed. We assume, since the snapshots were retrieved from the official system archives or version control systems, that the contents of the snapshots retrieved are the same. The only other metric provided by Bijlsma to verify this assumption is the size of the snapshot in LOC. Figure 2 compares the snapshot sizes as found by Bijlsma against the snapshot sizes found by us. It can be immediately seen that the blue and red graphs do not align. This behaviour is expected, however, as these numbers are retrieved *after* the SAT analysis.

The SAT requires a scoping definition per system in order to function. Scoping is the process of determining what parts and files of the system should be included in the calculation. For example, source files in a '/libs/' folder should not be included in the calculation as they are external dependencies and are not maintained by the development team directly. Ideally, the scoping per system should be exactly the same as Bijlsmas original scoping when rerunning the SAT. However, scoping files were not documented in Bijlsmas study, so we had to do our own scoping. Luckily, Bijlsma provided feedback in person to check if the scoping files were roughly the same.

Given the scoping differences, a slight deviation in SAT considered lines of code for the calculation can be explained. We do expect, however, that the newly acquired snapshots follow the same trend line as the old snapshots. For the majority of the systems listed in Figure 2 this is the case (e.g. abiword, ant, argouml, checkstyle), but some other systems stand out. Webkit, for example, only has a third of the original size (in KLOC). The cause for this large difference remains unknown to us, as running the SAT with all junk files included doesn't come near the originally reported size numbers. As such, the newly measured Webkit maintainability numbers can not be compared against the old, and will impact the correlation results.

System	Main Language	LOC (Latest) Replication	LOC (Latest) Original	Snapshots
Abiword	C++	427,885	415,647	3
Ant	Java	101,647	122,130	20
ArgoUML	Java	155,559	171,402	20
Checkstyle	Java	38,685	38,654	22
Hibernate-core	Java	114,953	145,482	5
JEdit	Java	109,403	100,159	4
Spring-framework	Java	98,141	118,833	21
Subversion	C	192,620	218,611	5
Tomcat	Java	158,881	163,589	19
Webkit	C++	425,929	1,255,543	1
Total				120

Table 4: Systems assessed by Bijlsma.

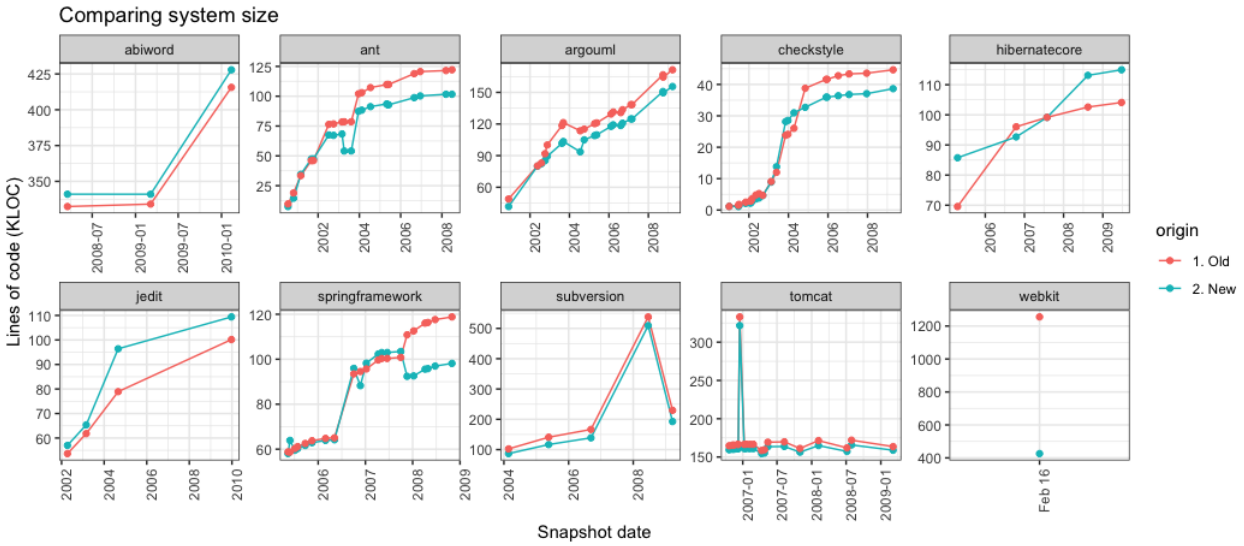

Figure 2: Comparing system sizes per snapshot (KLOC). Each point represents a snapshot of the given system, ordered by increasing date. Each snapshot is represented by two points, size in KLOC as found by Bijlsma (Red) and us (Blue).

3.2 Method Differences

To summarize, some elements of the original study's method have been kept exactly the same while other elements have been changed.

Differences

Since the original snapshots were not archived, the snapshots had to be reacquired. This results in small data inconsistencies for most systems and for large inconcistencies for one systems in particular (Webkit). Additionally, as is the purpose of this study, the *new* SIG maintainability model is used to measure maintainability as opposed to the original SIG maintainability model dating from 2010.

Equalities

Given the concept of issue resolution time and issue resolution time quality ratings, these ratings for all snapshots and systems have been re-used directly. The respective issue tracking systems were not mined again. Furthermore, the correlations are calculated in the same manner.

4 Results

4.1 Comparing Maintainability

Maintainability ratings are directly compared between the old and the new model to gain a better understanding to how the SIG maintainability model evolved. It also serves as a validation step to see if the new main-

tainability ratings are reasonable within expectation compared to the old ones. Figure 3 gives a high level overview on how the maintainability ratings per system (distributed over all snapshots) compare between the old and the new SIG maintainability model. We observe for the majority of the systems (ant through tomcat) new maintainability ratings are lower compared the the old model. This behaviour is expected because the SAT uses benchmark data to determine the thresholds of the rating buckets and has been rising over the years (See Section 5 for further elaboration). Note that, even though plotted, maintainability cannot be compared for webkit as the reacquired snapshot has over 500 KLOC less than the original.

The maintainability rating calculated by the SIG maintainability model is composed by a double aggregation. Figure 3 adds another aggregation on top of this, combining multiple maintainability ratings per system into a single boxplot. The figure provides a high level overview, but in order to discover the other factors that cause the deviation in maintainability ratings we need to zoom in on the low level metrics. Figure 4 illustrates all unit complexity ratings of all systems ordered by snapshot date. The figure shows how in general the new ratings follow the same trend as the old ratings, but on a slightly lower rating altogether. Specifically the systems ant, jedit, tomcat and springframework show this behaviour well. The lower rating can again be explained by the rising benchmark thresholds. Webkit consistently rates higher for all system metrics, but are insignificant due to the large deviation in reacquired snapshot sizes. ArgoUML consistently shows lower ratings for the original system properties (without the modularity system properties), but shows a higher rating in overall maintainability (Figure 3).

4.2 Comparing Correlations

Bijlsma and Luijting classified four types of issues: defect, enhancement, patch and task. Bijlsma and Luijten investigated issues of type defect and enhancement. Tables 5 and 6 illustrate the new correlations found for these two types of issues. Every correlation is tested for significance, given the following set of hypotheses $H_0\{\rho = 0\}$ and $H_A\{\rho > 0\}$. For the zero hypothesis to be rejected, a confidence threshold of 5% is used.

Given the new defect correlations, the correlations of the original system properties are comparable, except for module coupling which shows a significant drop from 0.55 to 0.36. The other surprising result is the large drop of maintainability from 0.64 to 0.33. The negative correlation of modularity is surprising, as it goes against our intuition. Intuitively, modular systems should be easier to modify than systems with huge, monolithic, components. Further, unit interfacing has vastly decreased in significance, from 0.042 towards 0.640.

Table 6 shows the same comparison of correlations as Table 5, but for enhancement resolution speed. The difference in maintainability correlations is a lot smaller compared to the difference found in the defect correlations. The modularity correlations also stand out, since both the coefficients of modularity and component balance cannot be assumed given their p-values being larger than 0.05. The decrease in significance is specifically interesting compared to the defect correlations in Table 5.

5 Discussion

5.1 Comparing Maintainability

One explanation for the lower maintainability is the calibration of benchmark thresholds to determine the ratings. Over the years new systems that are measured are added to the SAT benchmark. The observation is that the distribution of quality ratings in the benchmark, over time, shifts towards a higher average. In order to compensate for this phenomenon, SIG calibrates the thresholds for all ratings (both system-properties and characteristics) yearly. This means that the thresholds (for most of the characteristics) have become stricter. This is also documented in the 'Guidance for Producers' documents, which SIG releases yearly. For example, in their 2018 document it is mentioned for unit complexity that "To be eligible for certification at the level of 4 stars, for each programming language used the percentage of lines of code residing in units with McCabe complexity number higher than 5 should not exceed 21.1%" [Vis18] while their 2017 document states the same but with a threshold of 24.3% [Vis17]. Remeasuring the same systems again with the stricter benchmark thresholds results in overall lower maintainability scores.

This expected behaviour of lower maintainability ratings is consistent for eight out of ten systems. The systems Abiword and Webkit stand out as they both score higher compared to the original rating.

Webkit can be considered an outlier. The system is composed by a single snapshot that consistently scores higher for all system properties and aggregated ratings. This may be the result of the re-acquisition of the snapshot, as the newly obtained snapshot has roughly 500 KLOC less than documented by Bijlsma.

Abiword, however, does follow the expectations of lower ratings for system metrics. The overall higher maintainability score can be speculated by the new properties introduced in the new model (component balance and component independence). Specifically

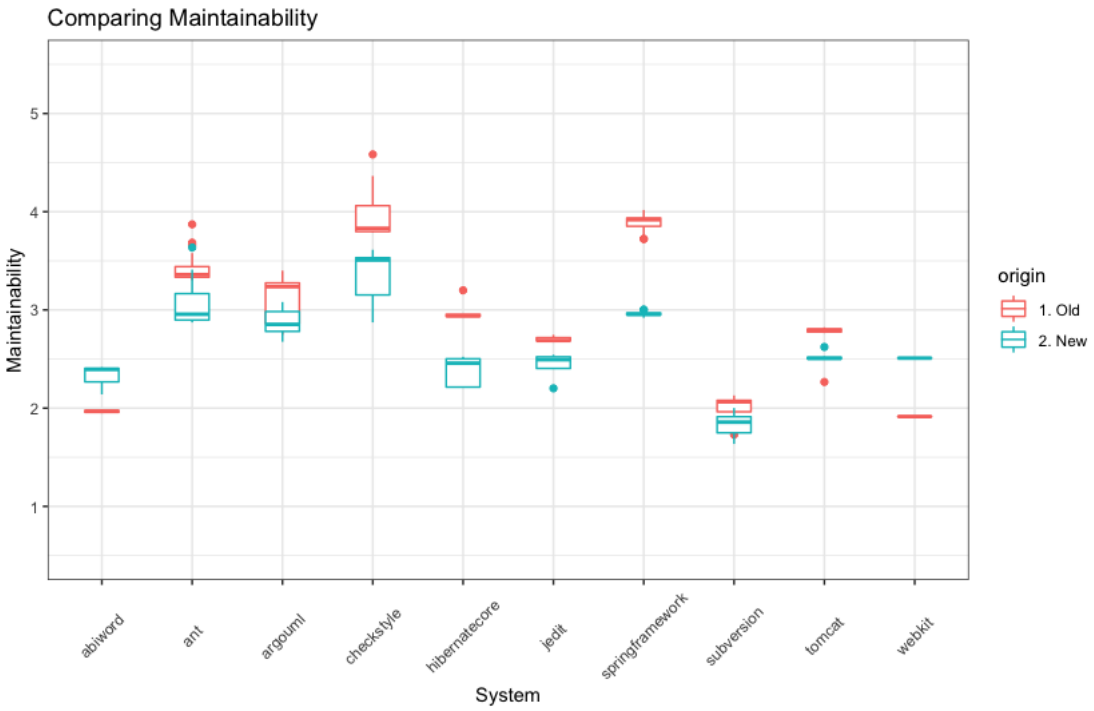

Figure 3: Snapshot maintainability distribution per system. Each system contains two boxplots, the maintainability ratings as obtained by Bijlsma (red) and the maintainability ratings obtained by the 2018 version of the SIG maintainability model (blue), distributed over all the snapshots of the system.

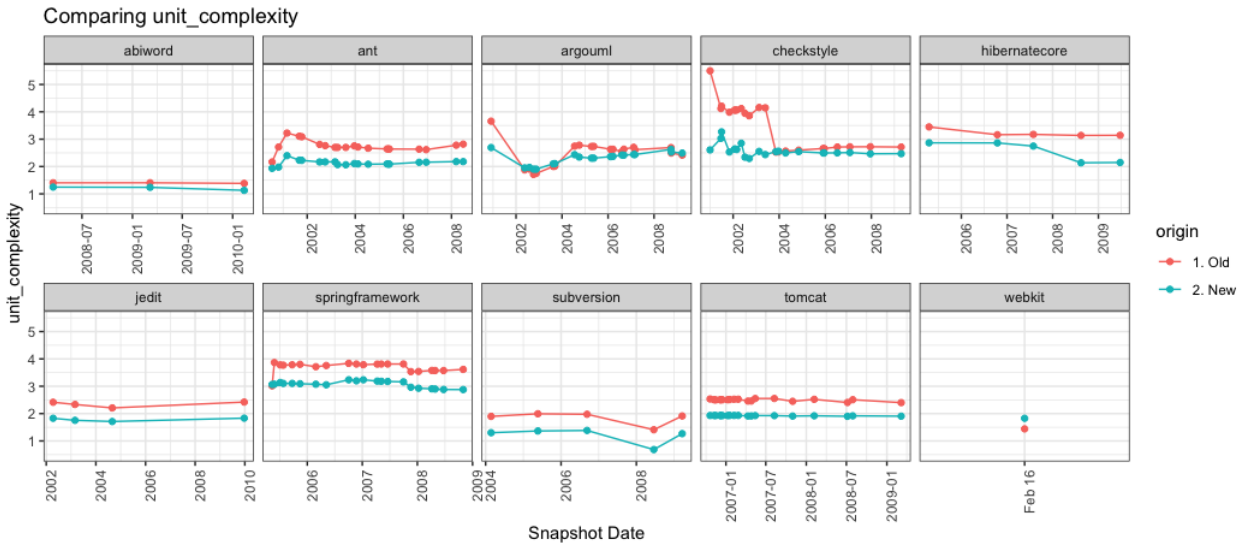

Figure 4: Unit complexity ratings per system. Each point represents a snapshot of the given system, ordered by increasing date. Each snapshot is represented by two points, unit complexity of the old model (red) and the new model (blue).

Table 5: Defect resolution time correlations. The Table on the left shows the correlation statistics found by Bijlsma. The Table on the right shows the correlation statistics as obtained by the replication study.

Old Correlations

Defect resolution vs.	ρ	*p-value*
Volume	0.33	0.001
Duplication	0.34	0.001
Unit size	0.53	0.000
Unit complexity	0.54	0.000
Unit interfacing	0.19	0.042
Module coupling	0.55	0.000
Analysability	0.57	0.000
Changeability	0.68	0.000
Stability	0.46	0.000
Testability	0.56	0.000
Maintainability	0.64	0.000

New Correlations

Defect resolution vs.	ρ	*p-value*
Volume	0.39	0.000
Duplication	0.38	0.000
Unit size	0.53	0.000
Unit complexity	0.50	0.000
Unit interfacing	0.05	0.640
Module coupling	0.36	0.000
Analyzability	0.33	0.002
Modifiability	0.59	0.000
Testability	0.49	0.000
Maintainability	0.33	0.001
Modularity	-0.30	0.004
Reusability	0.46	0.000
Component balance	-0.34	0.001
Component independence	0.16	0.201

Table 6: Enhancement resolution time correlations. The Table on the left shows the correlation statistics found by Bijlsma. The Table on the right shows the correlation statistics as obtained by the replication study.

Old Correlations

Enhancement resolution vs.	ρ	*p-value*
Volume	0.61	0.000
Duplication	0.02	0.448
Unit size	0.44	0.000
Unit complexity	0.48	0.000
Unit interfacing	0.10	0.213
Module coupling	0.69	0.000
Analysability	0.44	0.000
Changeability	0.46	0.000
Stability	0.50	0.000
Testability	0.47	0.000
Maintainability	0.53	0.000

New Correlations

Enhancement resolution vs.	ρ	*p-value*
Volume	0.58	0.000
Duplication	0.09	0.499
Unit size	0.45	0.000
Unit complexity	0.47	0.000
Unit interfacing	-0.20	0.132
Module coupling	0.67	0.000
Analyzability	0.22	0.096
Modifiability	0.52	0.000
Testability	0.68	0.000
Maintainability	0.47	0.000
Modularity	-0.09	0.513
Reusability	0.37	0.004
Component balance	-0.29	0.023
Component independence	0.34	0.039

because the component independence scores for the Abiword snapshots read 5.23, 5.23 and 2.50 ordered by date respectively.

5.2 Comparing Correlations

Since the original system properties are similar, it seems like the added maintainability sub-characteristic modularity with its system properties component balance and component independence are the biggest causing factor for the maintainability correlation to drop from 0.64 to 0.33. The negative correlation for modularity and component balance is surprising as it goes against our intuition. Overall one would assume a modular program would help defect and enhancement

issue resolution time instead of the opposite. However, perhaps the results make an argument for the way modularity is assessed currently. The performance of component balance, for example, has been debated before [BvDV13] (specifically, the discussion around the optimal number of components and the performance on smaller systems).

5.3 Threats to Validity

One of the main threats to validity is the variety in SAT scoping. In order to get accurate replication results, ideally, the scoping per system should be exactly the same as Bijlsma's original scoping when rerunning the SAT. As a consequence, results obtained may deviate slightly. However, given that the SIG maintainability model uses two level aggregation to compute the final maintainability score, small deviations in results should not affect the final maintainability score by a large margin.

An additional difference in scoping is the component_depth property, which was introduced when evolving according the new ISO 25010 standard (as described in section 2.1). This property needs to be set to show were the highest level components in the directory of a system reside. This is needed in order to calculate the modularity system properties. The ambiguity of the component definition requires an external validator to check for correctness. In our case, given the age of the systems, no external validator was approached to check if we defined the right highest level components. The component depth property was set in accordance with our own interpretation of the system.

6 Conclusion

In order to answer the research question: *What is the relation between software maintainability and issue resolution time?*, in this paper we provide answers to the sub-question: *"Does the previously found strong correlation between maintainability and issue resolution time still hold given the latest (2018) SIG maintainability model?"*. The experiment to find correlations between maintainability (as assessed by the SIG maintainability model) and issue resolution time, as originally defined and executed by Bijlsma and Luijten in 2012 [BFLV12] has been replicated. The experiment was run on the same, reacquired (with small deviations), snapshots of systems as in the original study with the *new* (2018) version of the SIG maintainability model.

Many similar correlations are observed between the 2010 and 2018 maintainability ratings versus the resolution time of defects and enhancements. However, regarding two new metrics in the 2018 model: (1) component balance does not correlate as expected, and (2) component independence correlates only in cases enhancements are considered.

Our next steps are to investigate the cause of the observed differences and further validate the underlying data. Additionally we would like to extend the data set to modern software systems.

7 Future Work

The system property component balance and its associated quality characteristic modularity can be considered a reason why the overall defect maintainability correlation is much lower than in the original study. Future work can expand in this direction, researching the effect of modularity on issue resolution time. Specifically, does the modularity coefficient look any different when the enhancement results are significant?

Next to expanding in the direction of modularity, more questions need to be answered in order to fully show the relation between maintainability and issue resolution time. Does the previously found relation still hold when tested against modern systems? Furthermore, Bijlsma analyzed mainly Java systems. How does this extend towards other languages? In this paper we tested against maintainability as assessed by the SIG maintainability model. However, in order to make the concept of maintainability more generalizable, do the correlations still hold when tested against other maintainability implementations (e.g. the maintainability index as proposed by Oman et al. [CALO94])?

References

[BFLV12] Dennis Bijlsma, Miguel Alexandre Ferreira, Bart Luijten, and Joost Visser. Faster issue resolution with higher technical quality of software. *Software quality journal*, 20(2):265–285, 2012.

[BHL+12] Tibor Bakota, Peter Hegedus, Gergely Ladányi, Peter Kortvelyesi, Rudolf Ferenc, and Tibor Gyimóthy. A cost model based on software maintainability. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 316–325. IEEE, 2012.

[BvDV13] Eric Bouwers, Arie van Deursen, and Joost Visser. Evaluating usefulness of software metrics: an industrial experience report. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 921–930. IEEE, 2013.

[CALO94] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

[HKV07] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *null*, pages 30–39. IEEE, 2007.

[ISO11a] ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. Standard, International Organization for Standardization, Geneva, CH, March 2011.

[ISO11b] ISO/IEC 25010:2011, Software engineering – Product quality – Part 1: Quality model. Standard, International Organization for Standardization, Geneva, CH, March 2011.

[Lui10] Bart Luijten. Faster defect resolution with higher technical quality of software. 2010.

[sig] Quality model 2018 announcement. www.softwareimprovementgroup.com/news-knowledge/sig-quality-model-2018-now-available/. Accessed: 2018-12-20.

[Vis17] Joost Visser. Sig/tüvit evaluation criteria trusted product maintainability: Guidance for producers. *Software Improvement Group, Tech. Rep.*, page 7, 2017.

[Vis18] Joost Visser. Sig/tüvit evaluation criteria trusted product maintainability: Guidance for producers. *Software Improvement Group, Tech. Rep.*, page 7, 2018.