The Correspondence between Software Quality Models and Technical Debt Estimation Approaches

Isaac Griffith, Derek Reimanis, Clemente Izurieta
Software Engineering Laboratory
Department of Computer Science
Montana State University
Bozeman, MT, 59717, USA
{isaac.griffith, derek.reimanis}@msu.montana.edu
clemente.izurieta@montana.edu

Zadia Codabux, Ajay Deo, Byron Williams
Department of Computer Science and Engineering
Mississippi State University
Starkville, MS, 39762, USA
{zc130, akd175}@msstate.edu
williams@cse.msstate.edu

***Abstract*—Technical debt has recently become a major concern in the software industry. While it has been shown that technical debt has an adverse effect on the quality of a software system, there has been little work to explore this relationship. Further, with the growing number of approaches to estimate the technical debt principal of a software system, there is a dearth of work to empirically validate the relationship between technical debt scores produced by practical tools against established theoretical quality models. We conducted a case study across 10 releases of 10 open source systems in order to evaluate three proposed methods of technical debt principal estimation. The evaluation compares each technique against an external quality model. We found that only one estimation technique had a strong correlation to the quality attributes reusability and understandability. In a multiple linear regression analysis we also found that a different estimation technique had a significant relationship to the quality attributes effectiveness and functionality. These results indicate that it is important that industry practitioners, ensure that the technical debt estimate they employ accurately depicts the effects of technical debt as viewed from their quality model.**

Keywords—technical debt, quality, empirical, model.

I. INTRODUCTION

The notion of technical debt [1] has become a fast growing phenomena in the area of Software Engineering. It encompasses a relationship between the decisions made and their effect on the quality of the system. Recently, several methods have been proposed to estimate a software system's level of technical debt [2] [3] [4] [5] [6] [7] [8] [9]. Unfortunately there has yet to be a study which evaluates whether these estimation techniques effectively describe the relationship between the quality of the system and the level of technical debt.

The need to measure the quality of a software product has existed nearly as long as software engineering itself [10]. This has produced several quality models and tools which have gained traction in industry. With the growing concern for technical debt and its lasting effects, it is apparent that there exists a need to connect the tools that have grown out of the technical debt management arena to existing quality models.

The interesting and overarching question then becomes: *What does the estimate of technical debt provided by approach X mean in the context of quality model Y?* In other words how can we evaluate the accuracy of technical debt estimation approaches in the context of an external quality model? Answering these questions will provide empirical evidence as to which approach is best suited for a software development organization.

The purpose of this study is to evaluate several technical debt estimation approaches in the context of an external quality model, and to determine if the relationships which are present in current models of technical debt agree with the quality models. Specifically, we evaluated the method embedded in the SonarQube™[1] TD-Plugin [4], CAST's method of technical debt estimation identified by Curtis, Sappidi, and Szynkarski [2][3], and Marinescu's method of technical debt estimation using design disharmonies [7] all methods were evaluated against the QMOOD quality model [11].

Using the Goal Question Metric paradigm [12] we have identified the following research goals (RG), their associated questions (RQ) and their associated metrics (M):

RG1. Evaluate how the SonarQube TD-Plugin method, the CAST method, and Marinescu's method for technical debt estimation approaches are related to quality.

- **RQ1.1.** What is the strength of the relationship between the technical debt estimates and quality attributes?
- **RQ1.2.** How does the strength of each relationship compare?

RG2. For each method of estimation identify how the technical debt and quality relationship is portrayed.

- **RQ2.1.** What is the estimated effect of a change in the technical debt estimate on each of the quality attributes?

M1. Correlation: a measure of the strength of the linear relationship between two variables [13].

M2. Technical Debt Estimate (TDE): an estimate associated with the value of the debt (in monetary terms) or with the cost associated in removing the debt (via

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

refactoring) in either monetary amounts or amount of effort (i.e., man-hours).

M3. Quality Estimate: This is an estimate of one of the following quality aspects (as defined in the QMOOD quality model): reusability, flexibility, understandability, functionality, extendibility, and effectiveness.

In this paper we address a gap in current research addressing the appropriateness of several technical debt estimates in representing the relationship between external quality models and technical debt. Each selected method of technical debt estimation uses a different underlying quality model for which direct comparison was not available. Thus, we selected a separate quality model to which we can compare each measure against. We look at the relationship between technical debt estimation measures and the quality model, QMOOD (Quality Model for Object Oriented Designs) [14]. QMOOD uses a metrics suite; which is hierarchical in nature, and is based on the idea of using only metrics to indicate the quality level of software, such that the quality of the software can be evaluated early in the development lifecycle.

This paper is organized as follows: Section 2 identifies relevant background information and related work. Section 3 elaborates on the underlying research methods. Section 4 presents the results and associated analysis. Section 5 presents threats to validity and their mitigation. Finally, section 6 concludes with a summary of the findings and an outlook towards future work.

II. Background and Related Work

A. Technical Debt Principal Estimation

Although there are many proposed methods of technical debt estimation [2][3][4][5][6][7][8][9], in this study we are concerned with only three.

The first method is implemented in the SonarQube™ TD-Plugin [4]. In order to evaluate each system we used SonarQube™ sonar runner to analyze the source code of each system. After the analysis we used the TD-Plugin (with default settings) to extract the technical debt estimate for each release. This method uses the following formula to calculate the technical debt value [4]:

$$Debt = duplication + violations + comments + coverage + complexity + design \quad (1)$$

$$duplication = cost\_to\_fix\_one\_block * duplicated\_blocks \quad (2)$$

$$violations = cost\_to\_fix\_one\_violation * mandatory\_violations \quad (3)$$

$$comments = cost\_to\_comment\_one\_API * public\_undocumented\_API \quad (4)$$

$$coverage = cost\_to\_cover\_one\_of\_complexity * uncovered\_complexity\_by\_tests \quad (5)$$

$$design = cost\_to\_cut\_an\_edge\_between\_two\_files * package\_edges\_weight \quad (6)$$

$$complexity = cost\_to\_split\_a\_method * (function\_complexity\_distribution \geq 8) + cost\_to\_split\_a\_class * (class\_complexity\_distribution \geq 60) \quad (7)$$

Where *duplication, violations, comments, coverage, complexity* and *cycles* secondary formulas is each measured in man-days. Each of the costs used in the secondary formulas can be set as parameters. We used the default values as described by Table 1. *Duplication* refers to the estimated effort associated with the removal of duplicated code in the system. *Violations* is the estimated effort associated with the removal of violations in the system. *Coverages* represents the estimated effort required to bring test coverage up to 80%. *Complexity* is the total estimated effort required to split every method and every class (of those requiring such a split). *Comments* refers to the estimated effort associated with documenting the undocumented portions of the API. *Design* refers to the estimated effort associated with cutting all existing edges between files. Each of the cost (estimated effort) (Table 1) are defined in man-hours, in order to convert this to man-days for the debt calculation, the default value of 8 hours per day is used. A final calculation is then performed to evaluate the cost per man-day of technical debt using a default value of $500.

The second method proposed by Curtis, Sappidi, and Szynkarski [2][3] estimates technical debt principal using a cost model based on detected violations. This method uses estimates of *time to fix* and *cost to fix* in order to connect these identified violations to a monetary value. The following equation is proposed as a means to measure the technical debt principal:

$$\begin{aligned} TDE = &(\Sigma HS * \%HS * \overline{HS_F} * HS_{cost}) \\ &+ (\Sigma MS * \%MS * \overline{MS_F} * MS_{cost}) \\ &+ (\Sigma LS * \%LS * \overline{LS_F} * LS_{cost}) \end{aligned} \quad (8)$$

Where ΣHS, ΣMS, and ΣLS are the count of *high severity*, *medium severity*, and *low severity* violations respectively. The values for *%HS*, *%MS*, and *%LS* represent the percentages of high, medium, and low severity violations intended to be fixed. The values of $\overline{HS_F}$, $\overline{MS_F}$, and $\overline{LS_F}$ represent the average time (in hours) required to fix per instance of each severity level. Finally, the values of HS_{cost}, MS_{cost}, and LS_{cost} represent the cost in monetary value per hour to perform the work. Curtis, Sappidi, and Szynkarski, provide three estimates for technical debt (see Table 2).

TABLE 1. Default cost values used in the calculation of technical debt in the SonarQube TD-Plugin [4].

Cost	Default Value (in man-hours)
cost_to_fix_one_block	2
cost_to_fix_one_violation	0.1
cost_to_comment_one_API	0.2
cost_to_cover_one_of_complexity	0.2
cost_to_split_a_method	0.5
cost_to_split_a_class	8
cost_to_cut_an_edge_between_two_files	4

TABLE 2. VALUES FOR ESTIMATES OF TDE AS PROPOSED BY CURTIS, SIPPIDI, AND SZYNKARSKI [3].

	Violation Severity Level	Estimate 1	Estimate 2	Estimate 3
Percent of Violations to be Fixed	High Severity	50%	100%	100%
	Medium Severity	25%	50%	–
	Low Severity	10%	–	–
Hours to Fix	High Severity	1hr	2.5 hrs	10% - 1 hr 20% - 2 hrs 40% - 4 hrs 15% - 6 hrs 10% - 8 hrs 5% - 16 hrs
	Medium Severity	1 hr	1 hr	–
	Low Severity	1 hr	–	–
Cost per Hour	All Severity Levels	$75	$75	$75

The final method for estimating technical debt was developed by Marinescu [7]. This method utilizes design disharmonies in the software to derive an index of the underlying issues in quality. Marinescu proposes that we measure the impact of these design disharmonies based on how they influence the underlying design, the level of granularity at which they manifest themselves (class or method) and the underlying severity of the disharmony based on the amount of code it impacts. . Here the influence, $I_{disharmony}$, is one of the following values: high=2.0, medium = 1.0, and low = 0.5. The granularity, $G_{disharmony}$, is either of the following values: method=1.0 or class=3.0. Finally the severity, $S_{instance}$, is based on how much a disharmony violates a given metrics threshold. The impact score of a given instance of a disharmony is calculated using the following formula [7]:

$$IS_{instance} = I_{disharmony} * G_{disharmony} * S_{instance} \quad (9)$$

Once the impact score is computed the overall debt sym ptoms index (DSI) can be evaluated using the following equation [7]:

$$DSI = \frac{\sum_{all\ instances} IS_{instance}}{KLOC} \quad (10)$$

Where KLOC is the number of thousands of lines of code for the software system under consideration. Marinescu indicates that this index value acts as a surrogate measure for the technical debt level of a software system.

B. Technical Debt and Quality

The relationship between technical debt and quality is not well understood. There has only been a small number of studies that have looked into this relationship. One of the first studies was conducted by Zazworka, Seaman, and Shull [14], where they confirmed that design debt, specifically God Classes, negatively impact quality. Zazworka, Shaw, Shull, and Seaman [15] further looked into the impact of maintainability on design debt, and confirmed that technical debt adversely affects the *maintainability* of software. Curtis, Sappidi, and Szynkarski [2][3], on the other hand, evaluated the impact of technical debt on the following quality aspects: *robustness*, *performance*, *security*, *changeability*, and *transferability*.

Each of these studies provides empirical evidence of the adverse effects of technical debt on software quality, but an empirical model relating the change in quality to the change in technical debt has yet to be developed. Further, with the exception of the study by Curtis, Sappidi, and Szynkarski [3], there are no studies evaluating the estimation techniques and their appropriateness in representing the relationship between external quality models and technical debt.

III. METHODS

A. Experimental Design

The experiment was conducted as a multiple case study across 10 open source software systems selected from the *Qualitas Corpus* [16]. Each system was selected using the following criteria:

1. Each system must have a minimum of 10 releases available for analysis.
2. Each release of a selected system must have a size of at least 25 KLOC but not more than 250 KLOC (due to license limitations of measurement tools).
3. Each system must be open source and implemented in the Java™[2] programming language.

Using this criteria we selected the open source software systems listed in Table 3.

In Section I we identified two research questions of interest for this study. These questions are addressed by the following hypotheses:

- $H_{1,1}$: There is a relationship between each technical debt estimate and each quality attribute
- $H_{2,1}$: There is a relationship between each technical debt estimate and each quality attribute when accounting for differences between systems.

[2] http://www.oracle.com/technetwork/java/index.html

TABLE 3. SELECTED OPEN SOURCE SOFTWARE SYSTEMS AND THEIR VERSIONS UNDER STUDY.

System	System Versions									
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Apache Commons Collections	1.0	2.0	2.1	2.1.1	3.0	3.1	3.2	3.2.1	4.0alpha1	4.0
FindBugs	1.2.1	1.3.4	1.3.5	1.3.6	1.3.7	1.3.8	1.3.9	2.0.0	2.0.3	3.0.0
JFreeChart	0.6.0	0.7.0	0.8.0	0.9.0	1.0.0	1.0.4	1.0.8	1.0.12	1.0.16	1.0.17
JHotDraw	5.2	5.3	5.4b2	6.0b1	7.0.6	7.1	7.2	7.4.1	7.5.1	7.6
JRuby	0.9.0	1.0	1.1	1.2.0	1.3.0	1.4.0	1.5.0	1.6.0	1.7.0	1.7.11
Apache Ant	1.5.2	1.5.4	1.6.0	1.6.2	1.6.4	1.7.0	1.8.0	1.8.3	1.9.0	1.9.3
JEdit	2.4final	3.0	3.0.1	3.2.2	4.0	4.1	4.2	4.3	4.5.0	5.1.0
Apache Cayenne	1.0.6	1.1	1.2	2.0.2	2.0.3	2.0.4	3.0	3.0.1	3.0.2	3.1M3
PMD	3.9	4.0	4.1	4.2	4.2.6	4.3	5.0.0	5.0.5	5.1.0	5.2.0
ProGuard	1.0	1.3	1.7.2	2.0.1	3.0.7	3.3	3.7	4.0	4.5	4.11

The first hypothesis was evaluated by measuring Kendall's Tau [17] correlation coefficient between each quality at tribute and each technical debt estimate. We selected Kendall's Tau statistic because it acts as a measure of the monotonicity of the relationship between two variables and is robust against non-linearity [13]. In other words, Kendall's Tau statistic provides a measure of the consistency of the trend between two variables.

To evaluate the second hypothesis we used a multiple linear regression analysis. This analysis method was selected in order to account for lurking variables (hidden variables which simultaneously affect two variables within a relationship and thus affects the relationship itself [13]) missed by the correlation analysis. To determine the extent of the relationship between each estimation technique and each quality attribute we used a multiple linear regression model. This model has one independent variable, the technical debt estimate with five levels (TDE1, TDE2a, TDE2b, TDE2c, and TDE3), and the dependent variable; which is one of the quality attributes (reusability, flexibility, undertandability, functionality, extendibility, and effectiveness). The model considered two blocking variables: lines of code as defined by Li and Henry [18] and the specific release of a system.

The experimental process is depicted in Fig. 1. Having selected the systems for study we begin by measuring each release of each system using the technical debt identification tools (see Fig. 1) while simultaneously extracting the necessary metrics using the tool Understand[3]. Understand was selected in due to the sheer number of metrics it calculates and for the ability to utilize its API to further extend its capabilities. This information is used to calculate (or extract) the technical debt estimates (see Section III.B) and the quality attribute estimates (see Section III.C). Once we have the estimated values we proceed to the analysis phase (see Section IV).

B. Technical Debt Estimation

As identified in Section II.A we selected three approaches for TDE measurement. Each of the following methods was selected due to their operationalization in existing tools. The first method, uses the SonarQube™ TD-Plugin [4], is fully automated and will be denoted as, *TDE1*. The second method is based on the work of Curtis, Sappidi, and Szynkarski [2][3], is semi-automatically calculated and will be as *TDE2a, TDE2b,* and *TDE2c*. The TDE2 estimates are measured by first collecting design flaws using the following tools: PMD [4] and FindBugs[5] (both via SonarQube™ sonar runner tools) across each system and its releases. These tools identify several levels of severity for rule violations they detect, which are then aggregated by SonarQube™. For this study we use the following conversion: High Severity = CRITICAL, Medium Severity = MAJOR, and Low Severity = MINOR. TDE2 is

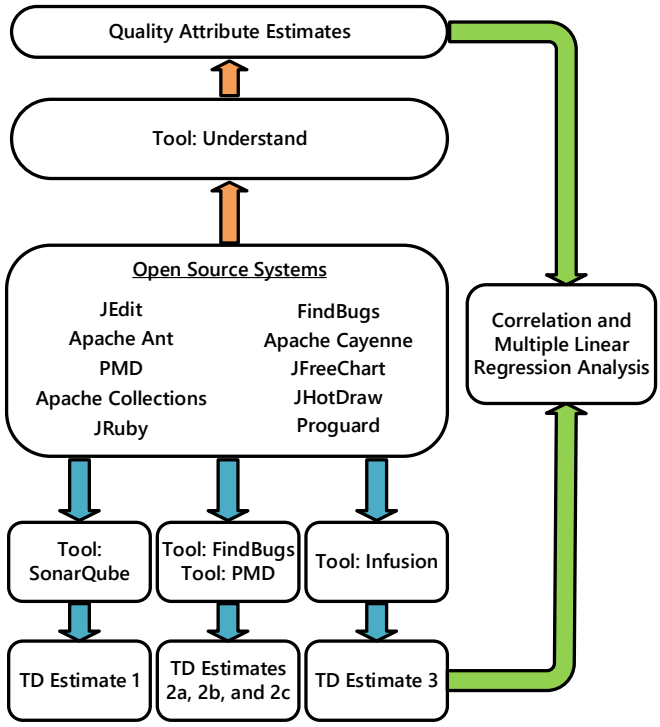

Fig. 1. Data collection process.

[3] http://www.scitools.com
[4] http://pmd.sourceforge.net/
[5] http://findbugs.sourceforge.net/

broken down into three separate estimates: $TDE2_A$, $TDE2_B$, and $TDE2_C$. Using the values provided in Table 1 in (8) we can calculate the estimates for each release of each system. The final method based on Marinescu's approach [7], is denoted as *TDE3*. TDE3 is automatically calculated as the *Quality Deficit Index* using the tool inFusion[6].

C. Quality Measurement

We use a third party quality model, QMOOD [11] to evaluate the accuracy of technical debt principal estimation techniques. QMOOD was selected due to its lack of relation to the underlying quality models used in each of the TDE measures, and it acts a representation of previous methods for quality assessment (the use of only metrics and the fact that it is based on the ISO 9126 specification [19]). Using an external quality model allows us to simulate the circumstances of a software development organization. The QMOOD quality model is based on the ISO 9126 specification [19] and uses of a combination of design metrics to indicate changes in system quality.

Each of the QMOOD quality aspects is measured using a combination of metrics as identified in [11] (see Table 4). The model is composed of the following six quality attributes:

TABLE 4. QMOOD METRIC MEASUREMENTS USING UNDERSTAND.

Metric	Measurement
Design Size (DSC)	Total number of classes defined in a system.
Hierarchies (NOH)	The count of the number of classes in a system where *MaxInheritanceTree(class) = 0*.
Abstraction (ANA)	The maximum length of a given class inheritance tree.
Encapsulation (DAM)	Ratio of the number of private and protected variables declared in a class to the number of total variables declared in the class.
Coupling (DCC)	A count of the number of couplings a class has.
Cohesion (CAM)	$\frac{100 - PercentLackOfCohesion}{100}$
Composition (MOA)	A count of the number of attributes which whose type is an object defined within the scope of the system.
Inheritance (MFA)	A count of the methods inherited by the class divided by the total number of methods available to the classs.
Polymorphism (NOP)	All methods excluding final, static, and private methods.
Messaging (CIS)	A count of the number of public methods declared in a class.
Complexity (NOM)	A count of all methods declared in a class.

TABLE 5. QMOOD QUALITY ATTRIBUTE EQUATIONS [7].

Quality Attribute	Calculation
Reusability	-0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 * Design Size
Flexibility	0.25 * Encapsulation – 0.25 * Coupling + 0.5 * Composition + 0.5 * Polymorphism
Understand-ability	-0.33 * Abstraction + 0.33 * Encapsulation – 0.33 * Coupling + 0.33 * Cohesion – 0.33 * Polymorphism – 0.33 * Complexity – 0.33 * Design Size
Functionality	0.12 * Cohesion + 0.22 * Polymorphism + 0.22 * Messaging + 0.22 * Design Size + 0.22 * Hierarchies
Extendibility	0.5 * Abstraction – 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Effectiveness	0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism

reusability, understandability, flexibility, effectiveness, functionality, and *extendibility.* The calculation of each of the quality attributes from the metrics listed in Table 4 is provided in Table 5. In order to measure these metrics we used the tool Understand. The QMOOD quality aspects and their relationships are provided in Table 5 (for convenience).

The measurement process of a given release of a system is as follows. First, we measure the metrics of concern (Table 4) for each release of each software system. We then calculate the values of the quality aspects using the metrics from the first step (see Table 5), for each release of each system respectively.

IV. RESULTS AND ANALYSIS

This section presents the results and discusses their analysis. The first set of results pertains to the evaluation of the relationship between the technical debt estimates and the quality attributes of the QMOOD quality model. The second set of results describes the multiple linear regression analysis.

We calculated Kendall's Tau correlation coefficient between each TDE and each quality attribute value as shown in Table 6 and Fig. 2. We tested for correlation between each paired sample. The test was conducted at the 95% confidence level. Those correlations with weak evidence (p-values < 0.1 indicate that there is a less than 10% chance that the observed value occurred by chance) are shown in bold in Table 6. The associated scatterplots for the correlations are displayed in Fig. 2. Fig. 2 can be read by finding the pair of variables along the diagonal finding either the scatterplot (below the diagonal) or

[6] http://www.intooitus.com/products/infusion

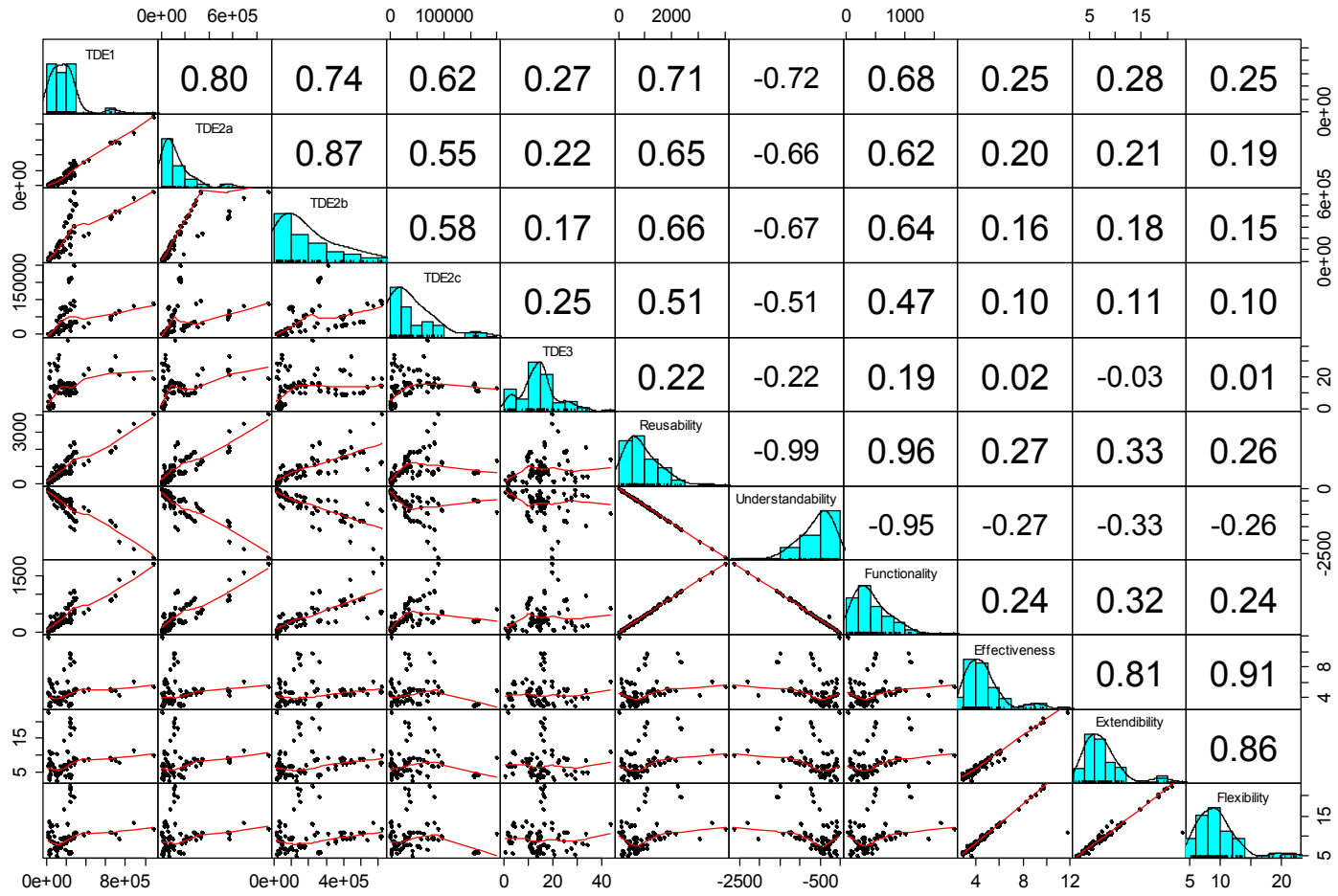

Fig. 2. Scatterplot and correlation matrix for TDE1, TDE2a, TDE2b, TDE2c, TDE3, Reusability, Understandability, Functionality, Effectiveness, Extendibility, and Flexibility.

correlation value (above the diagonal) where the rows and columns of the variables intersect.

As can be seen in Table 6, in all cases TDE3 shows weak correlation (< 0.45) (or no significant correlation) to each of the quality attributes. For reusability, understandability, and functionality there is moderate to strong correlation to TDE1, TDE2a, TDE2b, and TDE2c as shown in Table 6. Although these results are somewhat promising, they do not take into account the differences in size between the different systems nor the changes in size between releases of a system. To

TABLE 6. CORRELATION BETWEEN TD ESTIMATES AND QUALITY ATTRIBUTES.

	Technical Debt Estimates				
Quality	*TDE1*	*TDE2a*	*TDE2b*	*TDE2c*	*TDE3*
Reusability	0.7146	0.6483	0.6636	0.5059	0.2206
Flexibility	0.2538	0.1908	0.1481	**0.0955**	**0.012**
Understandability	-0.715	-0.658	-0.673	-0.506	-0.219
Effectiveness	0.2522	0.1964	0.1554	**0.0988**	**0.0156**
Functionality	0.6846	0.6175	0.6363	0.4748	0.1948
Extendibility	0.2805	0.211	0.1805	0.1105	**-0.03**

TABLE 7. INDICATION OF A RELATIONSHIP BETWEEN EACH OF THE TECHNICAL DEBT ESTIMATES AND EACH OF THE QMOOD QUALITY ATTRIBUTES. AN X INDICATES NO RELATIONSHIP AND A CHECK INDICATES A RELATIONSHIP.

	Technical Debt Estimate				
Quality	*TDE1*	*TDE2a*	*TDE2b*	*TDE2c*	*TDE3*
Reusability	✗	✗	✗	✗	✗
Flexibility	✗	✗	✗	✓	✗
Understandability	✗	✗	✗	✗	✗
Effectiveness	✗	✗	✗	✓	✗
Functionality	✗	✗	✗	✗	✗
Extendibility	✗	✗	✗	✗	✗

alleviate this threat, we also developed a multiple linear regression model which compensates for these issues.

The significance of the results from the multiple linear regression analysis are displayed in Table 7. For each of the technical debt estimation approaches we found that there was little to no evidence suggesting that the selected technical debt estimates have a relationship to reusability, understandability, functionality, and extendibility (as defined by the QMOOD quality model), while controlling for LOC and the number of releases in systems. With the exclusion of TDE2c, each of the remaining technical debt estimation techniques show little to no evidence of a relationship to flexibility or effectiveness.

There is strong evidence that for a one unit change in TDE2c there is a -1.701e-05 change in flexibility ($t_{87}=19.686$ with $p=0.000328$ and 95% CI between -2.60494e-05 and -7.973741e-06), when controlling for LOC and number of releases in systems. There is also strong evidence that for a one unit change in TDE2c there is a 1.508546e-13 change in effectiveness ($t_{87}=43.928$ with $p=0.001473$ and 95% CI between 2.352052e-14 and 3.886466e-13), when controlling for LOC and number of releases in systems.

In summary, as seen in Table 7, it appears that for all technical debt estimates excluding TDE2c they appear to have no relation to the QMOOD quality model, regardless of the correlation analysis shown in Table 6 and Fig. 2. We have demonstrated here, there is no evidence to suggest that these estimates of technical debt reflect the expected relationship to quality.

Software development organizations intent on utilizing technical debt estimation tools together with third party tools that measure a software product's quality, should ensure that the technical debt estimates are truly indicative of issues in quality of their product using their measurement system.

V. Threats to Validity

This section discusses potential threats to the validity of the study described within this paper. Specifically, we focus on threats to conclusion, internal, construct, content, and external validity [20][21].

A. Conclusion Validity

Conclusion validity is concerned with establishing statistical significance between the independent variables and the dependent variable outputs. A potential threat stems from the use of correlation analysis. Correlations do not provide a means to validate the statistical significance of the strength of relationships and is subject to issues with lurking variables. This threat is mitigated by the use of a regression model to further explore the relationships between the estimation approaches and quality attributes.

B. Internal Validity

This threat refers to the possibility of having unwanted or unanticipated relationships. If we had limited our analysis to only correlations between the technical debt estimates and quality attributes, then there would have been a threat to internal validity, but this threat has been mitigated in the multiple linear regression analysis through the use of a hierarchical model with a blocking variable on the size of the project.

C. Construct Validity

Construct validity refers to the meaningfulness of measurements and that the independent and dependent variables are represented correctly in the study. In our case study, the choice of the QMOOD model rather than any other quality model poses a threat. In addition, the metrics and thresholds for the quality attributes of the QMOOD model have not been evaluated to determine their appropriateness in all contexts. Another threat to the construct validity of this study is in the use of the tool Understand as a means to measure the QMOOD metrics. The QMOOD quality model was originally designed for the C++ language. In this study we looked at Java based projects and were forced to use a generic metrics tool to extract the necessary information. The threat stems from the potentially inaccurate interpretation of the metrics based on the descriptions provided by Bansiya and Davis [11] which is indicated as an existing issue by O'Keeffe and Cinneide [22]. A means to mitigate both of these threats would be to select more modern quality models for analysis. We leave this to future work.

D. Content Validity

Content validity refers to how complete the measures cover the content domain. QMOOD was based on the ISO 9126 specification [19], but it was created with the design phase in mind and so it does not take into consideration quality attributes related to implementation. Thus, we are not considering all attributes of quality which are relevant to the problem. Recent work by Ferenc, Hegedűs, and Gyimóthy [10] describes current software quality models and methods for comparison between these models. These more recent models incorporate not only software quality metrics but also software quality rules making such quality models more appropriate to the analyses performed across already implemented software systems. There is also a threat stemming from the selection of techniques for technical debt estimation. Although we have selected several techniques each one only estimates the technical debt principal, without providing measures for technical debt interest and interest probability [23]. This threat can mitigated by including estimation techniques that are more complete representations of technical debt, such as those from Nugroho, Vissier, and Kuipers [8] or Chin, Huddleston, Bodwell, and Gat [9].

E. External Validity

External Validity refers to the ability to generalize results. Due to the context of this case study, the ability to generalize from our results is limited. To make our results as generalizable as possible, we considered ten different Java open source systems from the *Qualitas Corpus* [16] and studied ten releases each. However, we cannot be sure our findings will be valid for other domains and applications.

VI. Conclusions and Future Work

We have conducted a case study to investigate the level of agreement from 5 methods of estimating technical debt principal to an external quality model. We studied 10 releases of 10 open source Java™ software systems. In order to evaluate whether the selected estimation approaches can be related to the attributes of the QMOOD quality model we conducted both a correlation analysis and a regression analysis.

Our initial correlation analysis found that there was some evidence for strong correlation between three of the estimates and reusability and understandability, which is expected given existing research. Unfortunately, correlation does not take other factors into consideration and to accommodate this we also performed a multiple linear regression analysis. The results of this latter analysis showed that with the exception of one estimation method (for flexibility and effectiveness) there was no observable relationship between the quality attributes and the technical debt estimates. Even more surprising was that given prior research showing that technical debt impacts both reusability and understandability of a software system, we found that for these quality attributes none of the technical debt principal estimates showed any relationship when taking size into consideration.

The method presented here of comparing technical debt estimates to the attributes of an external quality model can be used by practitioners and managers. It will provide them with an empirical assessment of selected technical debt measures against any quality model used to evaluate their software products. This will help assure that the technical debt values they are seeing are reflected as issues measured by their software quality model.

There are several paths for future work. In the short term exploring more recent quality models such as the SQUALE quality model [24], the QUAMOCO quality framework [25] or the Columbus quality model [26] against these estimates would be beneficial. Another immediate item is the evaluation of other methods of technical debt estimation, such as those put forth by Letouzey [6] and Letouzey and Ilkiewicz [5], Nugroho, Visser and Kuipers [8] and by Chin, Huddleston, Bodwell and Gat [9]. Along with exploring other existing methods of estimation, we need to develop a means to compare these methods and their merit in order to provide guidance to practitioners, similar to the work of Ferenc, Hegedűs, and Gyimóthy [10] on software product quality models. Lastly, this work explored these issues only within the context of open source Java™ software systems of moderate size. In order to validate these results we need to explore larger software systems (i.e., commercial software) as well as looking into software written in other languages.

References

[1] W. Cunningham, "The WyCash portfolio management system," *SIGPLAN OOPS Mess.,* vol. 4, no. 2, pp. 29-30, Dec 1992.

[2] B. Curtis, J. Sappidi and A. Szynkarski, "Estimating the Principal of an Application's Technical Debt," *Software, IEEE,* vol. 29, no. 6, pp. 34-42, Dec 2012.

[3] B. Curtis, J. Sappidi and A. Szynkarski, "Estimating the size, cost, and types of Technical Debt," in *Managing Technical Debt (MTD), 2012 Third International Workshop on*, 2012.

[4] O. Gaudin, "Evaluate your technical debt with Sonar," *Sonar, Jun,* 2009.

[5] J. Letouzey and M. Ilkiewicz, "Managing Technical Debt with the SQALE Method," *Software, IEEE,* vol. 29, no. 6, pp. 44-51, Dec 2012.

[6] J. Letouzey, "The SQALE method for evaluating Technical Debt," in *Managing Technical Debt (MTD), 2012 Third International Workshop on*, 2012.

[7] R. Marinescu, "Assessing technical debt by identifying design flaws in software systems," *IBM Journal of Research and Development,* vol. 56, no. 5, pp. 9:1--9:13, OCT 2012.

[8] A. Nugroho, J. Visser and T. Kuipers, "An empirical model of technical debt and interest," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, HI, USA, 2011.

[9] S. Chin, E. Huddleston, W. Bodwell and I. Gat, "The Economics of Technical Debt," *Cutter IT Journal,* vol. 23, no. 10, pp. 11-15, 2010.

[10] R. Ferenc, P. Hegedűs and T. Gyimóthy, "Software Product Quality Models," in *Evolving Software Systems*, T. Mens, A. Serebrenik and A. Cleve, Eds., Berlin, Heidelberg, Springer Berlin Heidelberg, 2014.

[11] J. Bansiya and C. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering,* vol. 28, no. 1, pp. 4-17, Jan 2002.

[12] V. R. Basili, "Software Modeling and Measurement: The Goal/Question/Metric Paradigm," College Park, MD, USA, 1992.

[13] R. D. De Veaux, Stats: data and models, 3rd ed ed., Boston: Pearson Education, 2012.

[14] N. Zazworka, C. Seaman and F. Shull, "Prioritizing design debt investment opportunities," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, HI, USA, 2011.

[15] N. Zazworka, M. A. Shaw, F. Shull and C. Seaman, "Investigating the impact of design debt on software quality," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, HI, USA, 2011.

[16] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton and J. Noble, "The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010.

[17] M. G. Kendall, "A New Measure of Rank Correlation," *Biometrika,* vol. 30, no. 1/2, p. 81, Jun 1938.

[18] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software,* vol. 23, no. 2, pp. 111-122, Nov 1993.

[19] *Software Product Evaluation--Quality Characteristics and Guidlines for Their Use,* ISO/IEC Standard ISO-9126, 1991.

[20] T. D. Cook and D. T. Campbell, Quasi-experimentation: design & analysis issues for field settings, Boston: Houghton Mifflin, 1979.

[21] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, Experimentation in Software Engineering, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[22] M. O'Keeffe and M. Cinneide, "Search-based software maintenance," 2006.

[23] S. McConnell, "Managing Technical Debt," 2008.

[24] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard and P. Vaillergues, "The squale model -- A practice-based industrial quality model," 2009.

[25] S. Wagner, K. Lochmann, L. Heinemann, M. Klas, A. Trendowicz, R. Plosch, A. Seidi, A. Goeb and J. Streit, "The Quamoco product quality modelling and assessment approach," 2012.

[26] T. Bakota, P. Hegedűs, P. Kortvelyesi, R. Ferenc and T. Gyimóthy, "A probabilistic software quality model," 2011.