An Industry Perspective to Comparing the SQALE and Quamoco Software Quality Models

Isaac Griffith, Clemente Izurieta
Software Engineering Laboratory
Montana State University
Bozeman, MT, USA
isaac.griffith@msu.montana.edu
clemente.izurieta@montana.edu

Chris Huvaere
TechLink Center
Software Engineering and Analysis Laboratory
Bozeman, MT, USA
chuvaere@techlinkcenter.org

***Abstract*—Context: We investigate the different perceptions of quality provided by leading operational quality models when used to evaluate software systems from an industry perspective. Goal: To compare and evaluate the quality assessments of two competing quality models and to develop an extensible solution to meet the quality assurance measurement needs of an industry stakeholder –The Construction Engineering Research Laboratory (CERL). Method: In cooperation with our industry partner TechLink, we operationalize the Quamoco quality model and employ a multiple case study design comparing the results of Quamoco and SQALE, two implementations of well known quality models. The study is conducted across current versions of several open source software projects sampled from GitHub and commercial software for sustainment management systems implemented in the C# language from our industry partner. Each project represents a separate embedded unit of study in a given context –open source or commercial. We employ inter-rater agreement and correlation analysis to compare the results of both models, focusing on Maintainability, Reliability, and Security assessments. Results: Our observations suggest that there is a significant disconnect between the assessments of quality under both quality models. Conclusion: In order to support industry adoption, additional work is required to bring competing implementations of quality models into alignment. This exploratory case study helps us shed light into this problem.**

***Index Terms*—quality assurance; quality standards; software quality**

I. Introduction

A theoretical quality model comprises a set of characteristics and sub characteristics that provide a platform on which quality assessments about software components can be made. By definition, theoretical quality models lack the ability to provide assessments of quality, thus their operationalization is necessary. The operationalization of these models is a critical step in providing pragmatic solutions that can be readily adopted by software development organizations in industry. Further, the deployment of operationalized quality models allow for continuous monitoring of the quality of an organization's software components; thus facilitating rapid intervention when violations are encountered.

In this multiple case study we operationalized the Quamoco [1] quality model in cooperation with our industry partner TechLink [2]. Further, we used our implementation to compare it with an existing and popular quality mode, SQALE [3], [4]. We have chosen to compare two implementations of ISO based quality models that focus not only on specific quality attributes, but also aggregate quality (from many dimensions; otherwise known as the *ilities* of software quality) to higher levels of abstraction. The connection to higher levels of abstraction help organization decision makers assess potential economic impacts of breakdowns in quality in a holistic manner. By focusing on the comparison between quality models with these characteristics, we facilitate an understanding of quality issues that affect decision makers as well as developers.

A. Industry Partners

We have two industry partners: i) TechLink, is a federally-funded technology transfer (T2) center that was established in Bozeman, Montana in FY 1996. In FY 1999, TechLink became the first DoD-wide Partnership Intermediary (per 15 USC 3715), with the mission to assist all branches of DoD with technology transfer, and ii) CERL [5] is the Construction Engineering Research Laboratory under the US Army Corps of Engineers, and the customer of this technology.

B. Research Objective

Our goal with this case study is to shed light into the inconsistencies with which quality is assessed. This is important because (pursuant with our research hypothesis) the perception of quality can differ significantly even when the underlying quality models are ISO based, and this helps generate confusion when industry practitioners such as CERL evaluate the quality of their software. We attribute these inconsistencies to the specific operationalizations of these theoretical models. To achieve this goal required the independent operationalization of a competing quality model –Quamoco (c.f. III-A).

C. Contributions

Our study provides the following four contributions: i) the operationalization of the Quamoco quality model as a plug-in to the SonarQube™ framework [6], which allows for easy comparison to the built-in SQALE method using the same rule sets (comprised of code smells, vulnerabilities, and bugs); ii) a comparison between two quality models and their respective

quality characteristics; which provides a much needed step towards the calibration of quality assessment reporting; iii) a comparison between open source and commercial grade software quality, and iv) an integrated visual dashboard that allows our industry stackeholder (i.e., CERL) and practitioners the ability to switch between quality models with a single click.

II. Background and Related Work

Quality models provide references that software components can be measured against. Theoretical models such as ISO specifications [7] [8] go only as far as offering guidelines along many dimensions of quality which must be operationalized to provide a working solution that can be used by the engineering community. A common criticism of theoretical models is that they are too ambiguous to be directly measurable. A comprehensive description of quality models is beyond the scope of this paper, however Wagner [9] and Ferenc et al. [10] provide a significant account and history of quality models. We selected two operationalizations of ISO theoretical models: SQALE and Quamoco.

In the last decade, the research community has also observed how technical debt has become a popular approach to track the progress of source code development by pointing out disharmonies (i.e. code smells) that need refactoring [11]–[13]. Their remediation can either be undertaken immediately, or scheduled for a later date at the expense of incurring debt (i.e. principal and interest) [14]–[17]. Tools are available that provide calculations of this index; however their reliability and the validity of their measurement methods remains an open and active research area. Technical debt should not be confused with software quality, as the former is a metric that only characterizes the maintainability of a system. The new definition of technical debt (16K definition) explicitly states that "*technical debt is a contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability.*" [18]. Although the focus of the definition is on only one aspect of the many dimensions that make up ISO based quality models, it is important to mention that the SQALE method to managing technical debt associates remediation costs that affect the technical debt index of a system by using a remediation function that takes into account all dimensions of quality, not just maintainability [19]. The SonarQube™ operationalization of SQALE deviates in its calculation of technical debt by only focusing on the technical debt ratio associated with maintainability. For a more comprehensive comparison between technical debt calculations and quality assessment approaches see Griffith et al. [20].

A. SQALE

The SQALE (Software Quality Assessment based on Lifecycle Expectations) quality model is a generic approach to modeling software quality and can be applied to any language. It is based on the ISO/IEC 9126-1:2001 standard [7] (further referred to as ISO 9K). The approach is based on eight code characteristics that are organized chronologically in pyramidal form. At the bottom of the pyramid is testability, followed above by reliability, changeability, efficiency, security, maintainability, portability and at the top of the pyramid, reusability. Quality requirements such as "Exception Handling shall not catch Null Pointer Exception," are associated with characteristics in the pyramid and have a remediation cost. If more that one characteristic is affected by a quality requirement then an association with the lowest characteristic is formed. The characteristics at the bottom of the pyramid represent more important dimensions of quality and are meant to aid practitioners when prioritizing requirements that need refactoring in the code base. SQALE is published under the open source licence and it is implemented by many vendors. This case study uses the implementation of SonarQube's plug-in as it has become widely adopted by organizations.

B. Quamoco

The Quamoco quality model was developed explicitly as an extensible meta model. Its goal was to bridge a gap between abstract concepts and measurable attributes. The central concept of the model is a factor, meant to represent an attribute or property of an entity; where the latter represents an important aspect of quality we want to measure. Two types of factors exist, quality aspects and product factors. The former represents the more abstract qualities found in theoretical models such as the ISO standards. The latter represents the measurable parts of a software component and have an impact on their associated quality aspect. Factors form hierarchies; where factors can further refine some aspect of quality. To improve modularization, the meta model is split into modules; where the root module contains general quality hierarchies and basic product factors. This allows practitioners to extend the root module for specific purposes or technologies, and to focus on the qualities relevant to their specific needs.

Quamoco's base model is an instantiation of the meta model and uses the ISO/IEC 25010:2011 [8] (further referred to as ISO 25K) as a reference. It is the result of many years of collaboration by quality experts from industry and academia, and it is comprised of a comprehensive set of factors and measures that capture software quality assessment.

C. High Level Differences between Quality Models

Quamoco and SQALE are both hierarchical models. They link issues found in software to quality aspects and sub-aspects. Both models use this information as a means to evaluate the quality of a software component. The more prominent differences between these models are:

i. Quamoco is defined using a meta-model which characterizes quality models defined for different circumstances. SQALE is limited to the model imposed.
ii. SQALE is limited to the aggregation of effort estimates through remediation functions. Quamoco is designed to incorporate weighted aggregation, ranking, and a variety of functions to describe the influence between aspects of the quality model.

iii. Quamoco models are defined in separate files and are hierarchical (in the sense that one model can inherit from another). The SQALE model is proprietary and built into the system.
iv. Quamoco models are defined such that an evaluation of the current level of quality can be provided without coaxing issues or rule violations into a unit-based measure. SQALE utilizes remediation effort as an index of quality, but its proprietary nature utilizes predefined values for each issue without the ability to easily tune or parameterize those values.
v. SQALE is based on ISO 9K, and Quamoco is based on ISO 25K.

III. Study Design

Careful consideration was given to the design of this study and we followed Yin's decompositions of case studies [21]. Although this study focuses on many subunits of study, their respective commercial and open source contexts go beyond a single holistic design. We gather data on each of the embedded units of analysis; however our larger exploratory analysis is the comparison between different quality models, and our focus is on understanding their differences and similarities among the various dimensions of quality.

The SQALE model is integrated with the SonarQube™ platform; however Quamoco required that we develop a plug-in to the SonarQube™ framework. By having both quality models integrated under a single platform, we reduce threats to the validity of the work, as the same environment needs to be configured for both models to be operational.

A. Quamoco Plug-In Architecture

Prior to comparing the assessments of both quality models, in cooperation with TechLink, we developed an extensible architecture to meet the functional needs of our industry stackeholder –CERL. The architecture that defines how our implementation of the Quamoco plugin[1] interacts with the SonarQube™ ecosystem is depicted in Fig. 1. The blue elements of the architecture represent the Quamoco plugin components embedded in SonarQube™, and the small yellow circles depict the order in which these components are executed. Critical components of this architecture include the following: i) the SonarQube™ Sonar Scanner Quamoco Sensor that executes the corresponding language parser to produce a code tree for a given source file and ii) the SonarQube™ Compute Engine; which executes the Quamoco Measure Computer.

The sensors are executed during the analysis phase by one of the many SonarQube™ scanner tools. In Step 1, the Sonar Scanner executes the SonarQube™C# analysis Sonar Plugin. This plugin identifies the source artifacts, collects issues from tools (such as Roslyn[2]) and calculates basic metrics at the file level and above. In Step 2 of the analysis phase, the Quamoco plugin executes, as follows: for each file identified in Step

[1]http://www.sparqline.com.
[2]https://github.com/dotnet/roslyn

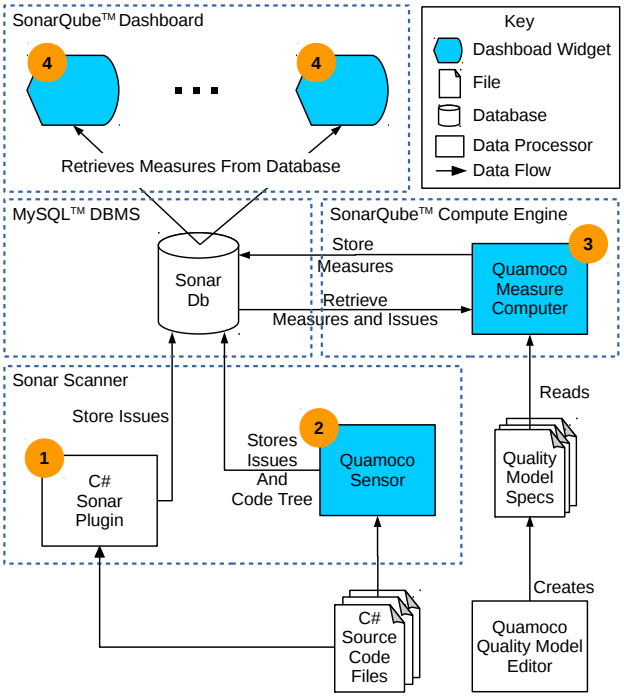

Fig. 1: Architecture diagram depicting the interaction of the Quamoco plugin and the SonarQube™ framework.

1, a code tree is generated and metrics for each node in the tree (excluding leaf nodes; which represent statements) are measured. Specifically the following metrics are measured: Number of Statements (NOS) [22], Source Lines of Code (SLOC) [22], Number of Fields (NOF) [23]–[25], Number of Methods (NOM) [26], and Number of Classes (NC) [27] for applicable levels. The values for these are then aggregated up the code tree and stored in each of the nodes. Once all files have been processed. The code tree is split into separate trees (corresponding to each file), converted to JSON and stored as a measure associated with the file in the analysis report for distribution to the server. This process is depicted in Fig. 2.

Once the analysis report has been received by the server the SonarQube™Compute Engine phase is executed, as depicted in Step 3 of Fig. 1. In this phase, artifacts are evaluated in a bottom-up fashion along the tree corresponding to the structure of the entire project. Starting with files, each file's code tree is extracted and merged to form the entire code tree used to evaluate quality. Issues identified by other plugins are also extracted. The code tree is used to identify the affected locations for potential issues (identifying methods, classes, or files) and this data is encoded into a *Finding*. At the module level code trees are joined into a sub-project node.

At the highest level of the tree (i.e., the root project level), the actual processing occurs. Any remaining modules are merged into the final code tree with the project at the root. Once this is complete, all metrics are aggregated to the top of the tree. At this point the system is ready to evaluate the quality of the project.

In order to evaluate the quality of a project, four activities

need to occur. First, the Quamoco quality model is opened and a processing graph is constructed. Second, the collected *Findings* are attached to the appropriate measures in the Quamoco processing graph. Third, quality is evaluated in a top-down recursive fashion (from the root "Quality" node down to each measure). Finally, the values of each quality attribute of concern are extracted and published in the SonarQube™ database. Each of these is further described in the following sub-subsections.

1) Processing Graph: The processing graph is a distillation of the combined quality models for a complete Quamoco instantiation for a given language. This graph is simply a directed acyclic graph composed of four types of nodes, as depicted in Fig. 3. Factor Nodes representing the higher level abstractions related to quality characteristics and subcharacteristics. Measure Nodes correspond to lower level issues related to the source code and which are applicable to entities found within source code (e.g., types, methods or fields). Finding and Value Nodes correspond to static analysis tool rules or metric values, respectively.

Each Factor Node has an attached evaluator which handles the evaluation of afferent (incoming) measures through finding the mean of the normalized value of the findings set or value set, or through a weighted sum of afferent factors. Measure nodes each have an attached aggregator operation applicable to the type of aggregation necessary: Union or Intersection for finding sets (propagated from attached finding nodes or other finding based measures) or Mean, Min, or Max for Value Nodes. Finding and Value Nodes provide the ability to collect either Findings (for named issues) or Values (for named metrics). Edges between Factor Nodes provide the necessary afferent weights (i.e., coefficients of source Factor Nodes) prior to summing the values at the destination Factor Node. Edges between Factor and Measure Nodes which convey sets of findings provide a means to normalize the finding set using an associated Normalization Measure and Range, along with a linearly increasing or decreasing function which constrains the value between 0.0 and 1.0, while also providing information on the expected effect that this measure has on the factor.

2) Collection of Findings: SonarQube™ utilizes both external and internal static analysis tools to provide a set of rules for a given language. Whenever these tools report a violation of one of these rules, SonarQube™ creates an Issue linked to the rule and the location within the File where the Issue occurred. Since SonarQube's smallest representable unit of a software system is the "File," and Quamoco requires that the representation occur at the finer-grained level of Types and Methods, this poses a problem. In order to solve this issue, our plugin also utilizes an ANTLR [3] parser to extract a representation of the system at this finer-grained level, and we call this representation a Code Tree. The Code Tree contains

[3]http://www.antlr.org

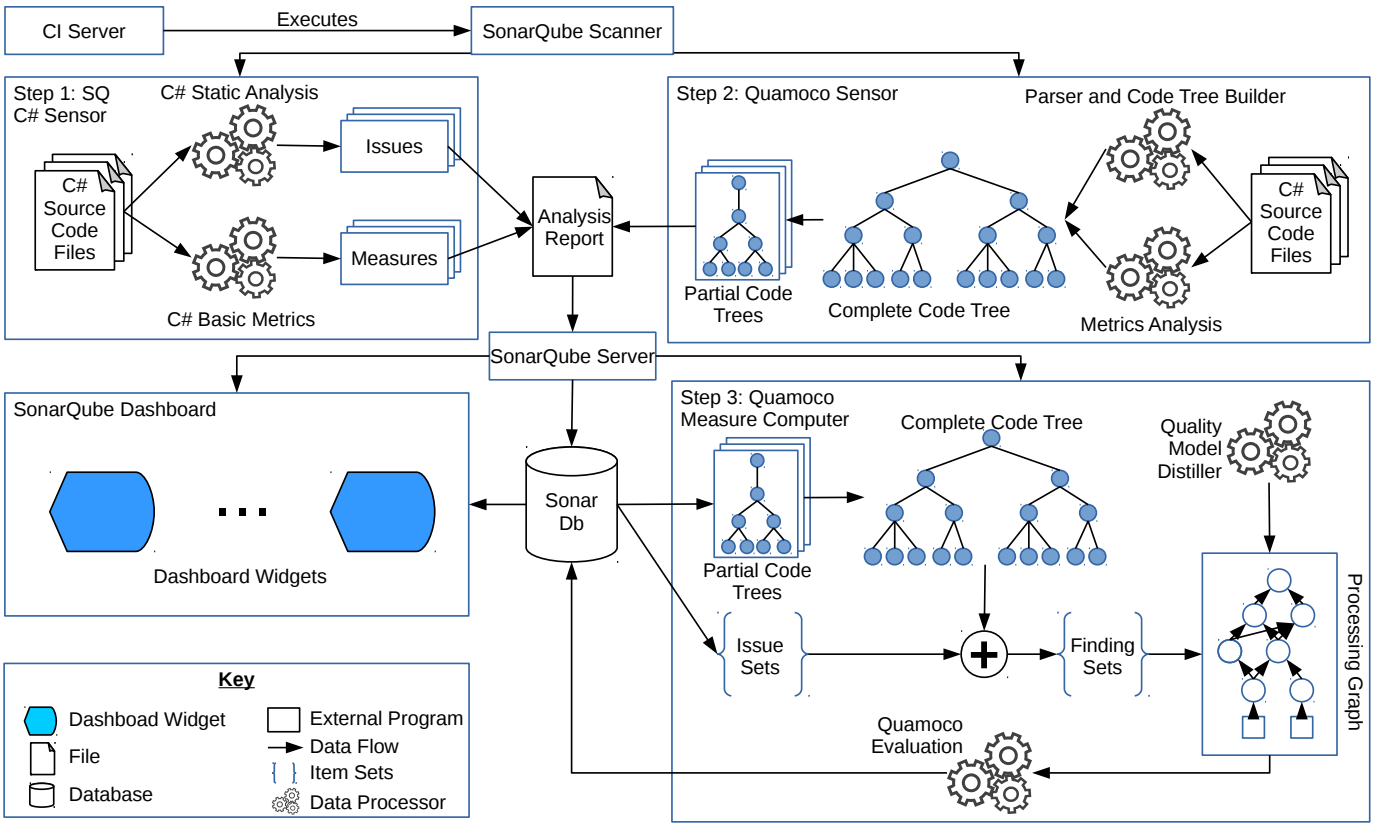

Fig. 2: Detailed architecture and flow process.

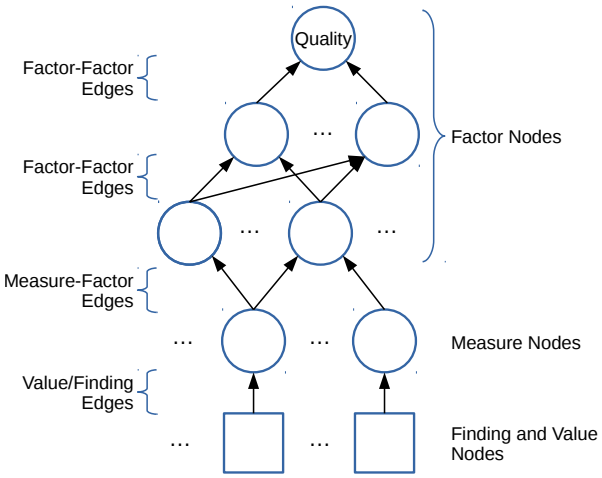

Fig. 3: Representation of the processing graph.

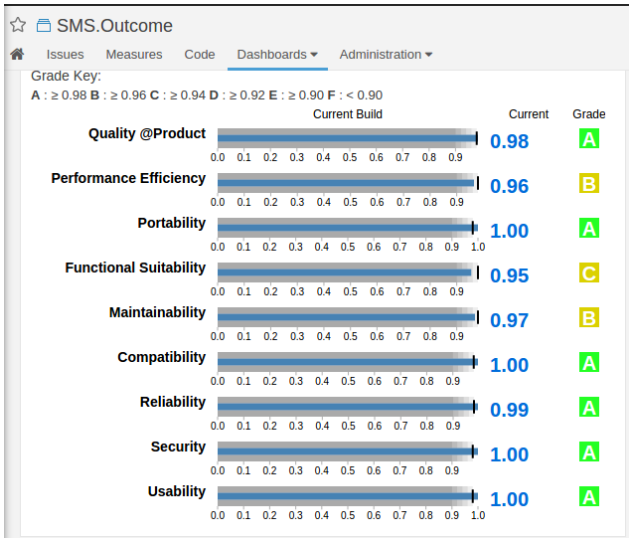

Fig. 4: Screenshot of the Quamoco Bullet Graph.

nodes from the Project-level down to the Statement-Level each of which are capable of containing metric information as well. Using this representation of the system the Quamoco plugin extracts the component where an issue is found (for each issue associated with each file) and constructs a Finding for that issue. The constructed finding contains the location (Method, Type, or File) where the issue was created and the name of the rule that was violated. Each created finding is then added to the Finding Node representing that rule in the Quamoco processing graph.

3) Evaluation of Quality: The Quamoco model evaluates the quality of a system by aggregating the measures and issues affecting the system. In a Quamoco model these form the lowest level of the hierarchy and provide input at the measure level of the model. Each measure can refine another measure or can be used in the evaluation of a factor. A factor can either be a combination of measures or a combination of factors, but not both. The value of a factor is always a value in the range [0.0, 1.0] and represents the presence of that factor within the software system. Measures which deal with issues simply pass sets of issues up to the next level of the hierarchy. Once the issue set reaches a factor they must be normalized.

An issue is normalized by summing the normalization measure (such as SLOC) across the entities (i.e., a method, class, or file) where the issue occurs and dividing this value by the total value of the metric across the system. This normalization then produces a value in the range [0.0, 1.0]. A factor which is evaluated by a set of other factors uses a weighted sum to determine its value. The weighted sum is defined by a rank assigned to each factor taking part in the evaluation, where higher ranks indicate lower importance. Weights are then assigned reflecting this ranking in order to ensure that the value produced is within the range noted above.

Controlling evaluation of the model uses a simple recursive depth-first search based algorithm. Starting at the sink Factor, "Quality", the algorithm requests the values for each incoming edge. This in turn requests the values of the source side node for each incoming factor, continuing down the factor hierarchy in the same manner. At the point when a measure to factor edge is reached it either requests the set of findings from the source or the set of values from the source (depending on the type of measure the source side is). This process stops once either a Finding Node or Value Node is reached. At this point set propagation or weight application along the edges and value aggregation or evaluation at the nodes occurs back up the graph (following the direction of the edges). It should be noted that once a value is calculated any subsequent calls on that node immediately return that value (thus reducing the number of full traversals of the graph).

4) Publishing Quality Information: Saving the quality values for a project is a simple matter. This occurs at the point when the project (top level component in the SonarQube™ system) is reached during the Compute Engine phase. Here the evaluation of the model is commenced. Once the value of the sink "Quality" factor has been determined, it and its immediate children's (representing the ISO 25K *illities*) values are collected and stored as measures on the Project component.

B. Dashboard Technology

The final step in generating quality ratings, is to extract the measured quality attributes from the SonarQube™ database and display the information using the dashboard technology as depicted in Fig. 4. In addition to the default SonarQube™ widgets, we have implemented a Quamoco bullet graph for displaying quality information. A bullet graph is the combination of a gradient scale, and a multi-level bar chart.

In Fig. 4, the background of each dimension of quality (i.e., a bar) is a gray gradient (with darker shades representing poor quality) that changes as the quality grade increases. Through the middle of each bar is a dark blue bar indicating the current level for a given quality attribute. Associated with each attribute is a scale between 0.0 and 1.0, indicating the normalized level of quality, and a marker depicted as a solid vertical black mark that represents the minimum threshold

TABLE I: Study projects and their basic properties. Note that with the exception of the systems in italics, all others are open source.

Project	Version	KLOC
DotNetOpenAuth	5.0.0	108.887
ElasticLINQ	1.4.1	4.682
EulerSharp	1.0	0.345
FireSharp	2.0.3	1.533
FluentCommandLineParser	1.2	1.890
GitVersion	4.0.0	7.779
LibuvSharp	1.0	7.973
MailChimp.NET	1.0	7.062
MongoRepository	1.6.11	1.503
NAnt	0.92	41.116
Prism	6.1.0	8.11
PythonNet	2.2.2	10.036
QRCoder	1.0	2.348
RemoveEmptyDirectories	2.2	1.740
SharpDox	1.2.1	8.535
SharpSSH	1.1.1.13	17.189
SMS Fueler	1.0	39.791
SMS Outcome	1.0	6.907
SpotifyAPI.NET	2.13.1	4.129
SurgarRestSharp	1.0.0	3.322

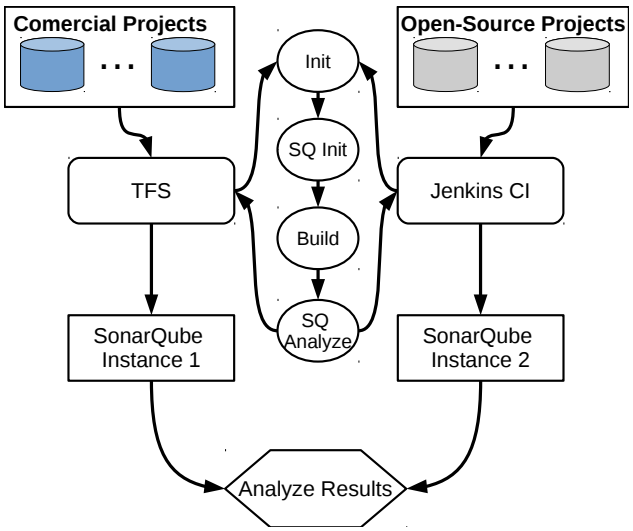

Fig. 5: The analysis process for both CERL and open source systems analyzed in this study.

necessary to obtain an A rating in the corresponding quality dimension. Currently this threshold is set at 0.98 (see Section III-D for a description of mapping Quamoco grade ratings) for all dimensions of quality. On the rightmost margin of the dashboard you can see the grade (in letter form A-F) that represents the rating of each quality dimension measured by Quamoco.

The ratings for the SQALE quality dimensions are provided by dashboard technology that is built in with the SonarQube™ tool. Since we use the open source version of the SonarQube™ tool, we only have access to a limited subset of quality attributes (c.f. III-D).

C. Case Studies and Context

We analyze the quality of CERL and open source software systems. CERL projects are from the unclassified Defense Logistics Agency (DLA) suite of Sustainment Management System (SMS) applications known as SMS Outcome and SMS Fueler, both of which are written in C#. The open source projects are a random selection of GitHub[4] C# projects. The names of each project and their corresponding sizes (in KLOC) are provided in Table I.

D. Approach

This study utilizes automated build and analysis tools to collect relevant data. The build tools selected were Jenkins CI[5] (for open source projects) and Microsoft Team Foundation Server (TFS)[6] (for CERL/commercial projects), as they provide the ability to automate the extraction of source code from associated repositories and to automate the build and analysis steps. The analysis was conducted using SonarQube™ 5.6.5 coupled with the provided C# plugin, and our own quality analysis plugins. This process is depicted in Fig. 5.

The analysis of the results of the two quality models is slightly more complicated than the simplified data collection process. Specifically, due to the limitations of the open source version of SonarQube™ we were restricted to comparing quality characteristics common between the SonarQube™ implementation of SQALE and our implementation of Quamoco. The common quality characteristics available were the following: Maintainability, Reliability, and Security. Both SQALE and Quamoco derive an ordinal assessment of quality for each dimension measured. Our implementation of Quamoco uses the presence of a factor within the system to calculate a normalized value bounded in the range [0.0, 1.0].

Although the ISO 9K standard (operationalized by SQALE) does not include Security as a characteristic, SonarQube™ does provide a rating for Security. The SQALE model calculates the Reliability of a system by counting the number of bugs, where 0 bugs yields an A, at least 1 minor bug yields a B, at least 1 major bug yields a C, at least 1 critical bug yields a D, and at least 1 blocker bug yields an E. The Security rating of a system is calculated in a similar manner as the Reliability aspect, but instead of bugs, they count vulnerabilities. The Maintainability of a system is calculated as a function of the technical debt ratio; where the technical debt ratio is equal to the remediation cost divided by the development cost.

The Quamoco quality ratings are a more straight forward conversion from the value representations. As described in Section III-A3, values of Quamoco factors fall in the range [0.0, 1.0] and represent the presence of that factor within the software system. Given this, Quamoco quality ratings simply become a function of mapping a quality factor's value into a grade. The current implementation uses the default grading scheme defined by Wagner [9] as follows, where x is a factor's value: A: $x \geq 0.98$, B: $x \geq 0.96$, C: $x \geq 0.94$, D: $x \geq 0.92$,

[4]https://github.com
[5]https://jenkins.io
[6]https://www.visualstudio.com/tfs/

E: $x \geq 0.90$ and F: $x < 0.90$.

Although both models arrive at their respective ordinal assessments of quality using different techniques, their respective calculations are based on empirical data that are meant to capture significant experiences from a large corpus of systems; thus the expectation from an industry perspective (i.e., practitioners and quality assurance engineers) is that similar ratings of quality should exist.

Having identified a common set of quality characteristics and a common notion of quality via the grade-based ratings, we can now compare default perceptions of quality according to both quality models. The ordinal grade metric can be directly compared using a non-parametric approach. One small issue is that SQALE ratings range from A-E and Quamoco ratings range from A-F. To handle this issue we categorize an E in SQALE in the same category as an E and F in Quamoco.

E. Statistical Analysis Procedures

The research objective is to identify inconsistencies in the interpretation of software quality via operationalization of a standard into a quality model. This objective then becomes the problem of comparing two quality models across relevant quality characteristics using the ordinal grade ratings, as identified in the previous section. To help illuminate this problem we have selected the following statistics: Bangdiwala's B [28], Cohen's κ [29] and Goodman-Kruskal's gamma [30].

Bangdiwala's B is a lesser known measure of inter-rater agreement and typically used to visually depict agreement. It measures the proportion of agreement in the range [0.0, 1.0], where 0.0 indicates no agreement, and 1.0 indicates complete agreement. Bangdiwala's B assumes that the data can be formulated as a frequency table (so at least nominal). Furthermore, this measure produces two values, the first is an unweighted version measuring the proportion of exact agreement between raters, while a weighted version measures the proportion of partial agreement between raters. For each quality characteristic we evaluate both the weighted and unweighted values of the B statistic in order to gain a good sense of the underlying data.

Cohen's κ is a well known non-parametric measure of inter-rater agreement and assumes that the underlying data is nominal, thus allowing the data to be analyzed without any transformation. The values of κ fall in the range [0.0, 1.0], where 0.0 indicates no agreement and 1.0 indicates complete agreement. For each quality characteristic we evaluate the κ statistic and calculate its 95% confidence interval. Unlike the B statistic, the κ statistic takes into account the fact that agreement may occur by chance, making it a more robust measure of inter-rater agreement.

In addition to the inter-rater agreement measures we have also selected the Goodman-Kruskal γ measure of ranked correlation. We have selected this measure as it is a non-parametric measure of associativity. The base assumption of the Goodman-Kruskal measure is that the variables in question are measured at the ordinal level, an assumption which our data meets. Furthermore, this measure does not penalize values due to ties, such as in Kendall's tau-b [31] (a more commonly used non-parametric measure of correlation), as we expect that ties will be common for both Quamoco and SQALE. Like all measures of correlation, the Goodman-Kruskal γ measure produces a value in the range [-1.0, 1.0], here a value of -1.0 represents complete disagreement or reciprocal association, a value of 0.0 represents ambivalence or no association, and a value of 1.0 represents complete agreement or association. Thus, it serves as a secondary measure for validation. For each quality characteristic we evaluate the γ statistic and calculate its 95% confidence interval.

TABLE II: Results of quality analysis across each of the selected projects. Pairs marked in bold indicate agreement between quality models. Note: columns with Q are Quamoco resuls and S are SQALE results.

	Reliability		**Security**		**Maintainability**	
Project	**Q**	**S**	**Q**	**S**	**Q**	**S**
DotNetOpenAuth	D	E	A	D	F	A
ElasticLINQ	A	D	**A**	**A**	B	A
EulerSharp	**A**	**A**	A	D	**A**	**A**
FireSharp	A	C	**A**	**A**	**A**	**A**
FluentCommandLineP	A	D	**A**	**A**	B	A
GitVersion	A	E	A	D	C	A
LibuvSharp	A	E	A	D	F	A
MailChimp.NET	**A**	**A**	**A**	**A**	B	A
MongoRepository	**A**	**A**	A	A	C	A
NAnt	A	E	A	D	F	A
Prism	A	D	**A**	**A**	D	A
PythonNet	A	E	A	D	E	A
QRCoder	A	E	A	D	C	A
RemoveEmptyDirector	A	D	**A**	**A**	C	A
SharpDox	C	E	A	D	**A**	**A**
SharpSSH	D	E	A	D	F	B
SMS Fueler	A	E	**A**	**A**	F	A
SMS Outcome	A	D	**A**	**A**	C	A
SpotifyAPI.NET	A	D	A	C	C	A
SurgarRestSharp	A	E	**A**	**A**	C	A

IV. Results and Analysis

This section presents the results and discusses their analysis. The first set of results pertain to the quality ratings by both Quamoco and SQALE quality models for the selected quality characteristics (see Table II). Initial non-parametric results of running the Wilcoxon [32] test for paired dependent samples yielded significant results ($p < 0.01$) for each of the three quality characteristics. These results were not unexpected and easily validated through simple visual inspection of the data. Our analysis thus turned to inter-rater agreement investigations. The second set of results (see Tables III and IV) are from the inter-rater statistical analyses conducted on the collected quality assessments.

The first set of results include the quality ratings across the 20 open source and commercial projects under study (as identified in Table I). Note that the results are an ordinal grade rating in the range of A-E for SQALE and A-F for Quamoco.

To compare the resulting analysis in Table II, we used inter-rater agreement measures Bangdiwala's B (see Table III) and Cohen's κ (see Table IV). Both of these measures assume that the provided data is from a nominal scale, yet the data

TABLE III: Results of the Bangdiwala B inter-rater agreement analysis.

Quality Characteristic	Bangdiwala	
	Unweighted	Weighted
Maintainability	0.15	0.291
Reliability	0.141	0.141
Security	0.5	0.5

TABLE IV: Results of Cohen's κ and Goodman-Kruskal's γ analyses.

Quality Characteristic	Cohen		Goodman-Kruskal	
	κ	CI	γ	CI
Maintainability	0.0	(-0.014, 0.014)	0.032	(-0.275, 0.327)
Reliability	0.0	(-0.06, 0.036)	-0.074	(-0.3411, 0.194)
Security	0.0	(0.0, 0.0)	0.038	(-0.238, 0.358)

collected is on an ordinal scale. Additionally, the Bangdiwala B statistic provides both an unweighted and weighted measure indicating the proportion of exact and partial agreement, respectively. In order to utilize the added dimension of order, which is unused by the prior two measures, we have also used the Goodman-Kruskal γ measure of ranked correlation. Both the value of γ and its associated 95% confidence interval for each quality characteristic are shown in Table IV.

V. Discussion

Although we only examine two commercial systems: SMS Fueler and SMS Outcome, it is clear that with the exception of the Security dimension, there is significant disagreement in terms of the quality indices reported by either method. In fact, with regard to Reliability and Maintainability, the assessments are very different. Interestingly, whilst Quamoco rates Reliability much higher than SQALE, the exact opposite is observed for Maintainability. With regards to Reliability, SQALE is significantly less forgiving because it only focuses on the number of vulnerabilities found; whereas Quamoco takes into consideration other factors aggregated at a higher level of abstraction. In the Maintainability case, SQALE maps a technical debt index to a letter grade; whereas Quamoco aggregates many metrics into quality aspects. For the eighteen open source projects we also see little in terms of agreements for each of the quality assessments, and all of the agreements tend to occur when the measured characteristic has an A rating (as indicated by bold pairings in Table II). These observations are further confirmed by the Bangdiwala analysis.

The results of the Cohen's κ analysis strongly indicates there is no agreement between the Quamoco and SQALE models in any of the three quality characteristics, when taking the selected projects as a whole across each dimension. This is further enforced by the Goodman-Kruskal's γ analysis which shows little deviation from 0 for each quality characteristic, and furthermore the 95% confidence interval for each characteristic includes 0.0, indicating for each quality characteristic there is no association between the Quamoco and SQALE models.

The data, across all categories, suggests that there is little agreement between models (for this sample of projects), as indicated by both the kappa and gamma analyses. Yet, the Bangdiwala B analysis shows us there is some agreement for these 20 projects specifically when there are little to no major issues affecting the software (in other words an A grade). A takeaway from these two operationalizations of quality is that SQALE ad Quamoco tend to agree when there are few major issues, but when a large number of issues begin to affect a project these models handle this in exceptionally different ways. Thus, this is indicative that further attention is needed to improve these operationalizations to align their notion of these quality characteristics.

Although the calculation method for each of the quality characteristics is different between both operationalizations of the quality models, the lack of alignment in terms of a final quality rating along quality dimensions contributes to the confusion of practitioners. Both of these assessment quality models are operationalizations of theoretical (i.e. definitional) ISO standards. Quamoco implements ISO 25K, and SQALE implements ISO 9K. Both hierarchical and the former, meant to be an improvement over the later. Thus, software engineering practitioners and decision makers should expect similar ratings in quality. In this case study, this is not the case, further contributing to a confusing landscape.

VI. Threats to Validity

In this section we examine the threats to the validity of this study as described by Cook and Campbell [33], Campbell and Stanley [34], Wohlin et al. [35] and Yin [21]. We are concerned with conclusion, content, internal, construct, external and reliability threats to validity.

A. Conclusion Validity

This validity check is concerned with establishing statistically significant relationships between the independent and dependent variables. Due to the nature of this study and the fact that there is no dependent variable to which a relationship would need to be established, there are no threats from this perspective. Although our statistical analysis was limited to Bangdiawala's B and Cohen's κ measures of inter-rater agreement and Goodman-Kruskal γ association tests, their application does not indicate inaccurate causes to affect type I or type II errors.

B. Content Validity

Content validity refers to how complete the measures cover the content domain. The Quamoco implementation covers the entire product domain of software quality (as defined in the ISO 25K standard), but we were limited by the availability of SQALE measures in the open source version of SonarQube™. Thus, we were limited to only Reliability, Security, and Maintainability.

C. Internal Validity

This threat refers to the possibility of having unwanted or unanticipated causal relationships. Given that this was a multiple case study of software systems, in which we exercised no

controls there is the possibility of unwanted or unanticipated casual relationships. Specifically, there could be unwanted or unnecessary connections between factors within the Quamoco model. The SQALE model (as implemented in SonarQube™) may also contain unanticipated or inappropriate mappings of issues to categories used in the evaluation of Reliability, Security, or Maintainability. Both cases pose potential threats to the internal validity of this study. Although we did modify the Quamoco C# model (switching it from the Gendarme[7] tool to Roslyn) by adding in new issues, measures, and factors, the vast majority of the model remains unchanged, with the exception of the ranks in the evaluation of Factors of the model. The ranks were changed to reflect the ranking of issues within the SonarQube™ platform.

The cumulative effect of these changes on any one factor was simply to either add new items or the changing of ranks (from a value between 1 and 3 to a value between 1 and 5). The rank changes were mostly superficial and only in rare cases changed to weights associated with factors, on the other hand the addition of new items had a more direct impact on the weights. Overall, the inclusion of new items in the Quamoco model does not invalidate it but rather calls for validation by experts in each of the affected quality attributes, which would help mitigate this threat. The SQALE model on the other hand is provided the effects of each issue via plugins to the SonarQube™ framework. Unfortunately we cannot validate or change these values which leaves no path for mitigation of this threat.

D. Construct Validity

Construct validity refers to the meaningfulness of measurements and the quality choices made about independent and dependent variables such that these variables are representative of the theory. The calculations for the SQALE quality indices, as implemented in the SonarQube™ framework, are trusted to represent empirical observations. The threat here is the question of what these values mean within the context of the software under measure. Thus, there is a potential threat to construct validity due to the implementation of the SQALE model.

Similarly, in the Quamoco model the grade values are determined by a threshold (as described in Section III-D), and by a weighting arrangement derived from trusted experiences of the original designers.

E. External Validity

External Validity refers to the ability to generalize results. In this study, as in most case studies, the ability to generalize results is limited due to a lack of ability to randomly sample or to randomly assign subjects to groups. In the case of this study, it was an observational study with the group (commercial or open-source) being an inherent trait of each system. Thus our results are limited, statistically, to the specific cases we studied.

[7]http://www.mono-project.com/docs/tools+libraries/tools/gendarme/

F. Reliabilty

Reliability refers to how dependent the process is on the researcher [21]. The process governing the quality evaluations are automated using Jenkins CI (for open source systems) or Microsoft Team Foundation Server (for CERL) and SonarQube™ and is not left up to the devices of any one researcher (with the exception of the interpretation of the results). For these reasons there are no unmitigated threats to reliability of the experiments.

VII. Conclusions and Future Work

We set out to explore and compare two operationalizations of well known theoretical quality models. To do this required the operationalization of the Quamoco quality model. Our research hypothesis (i.e., a statement of how we think quality assessments are implemented) describes our suspicion that different quality model implementations would likely report significant differences when assessing the same software components. As expected, we find disagreements between the ratings of software systems. Our early exploratory results indicate this is the case in both open source and commercial systems as suspected by TechLink, our industry partner, and by our research laboratory. Unfortunately this is a problem in an industry where there is little agreement in terms of implementing these models. Although hierarchical definitional (theoretical) models do agree on the characteristics and sub characteristics, there is still significant work that needs to be carried out to bridge the gap that operationalizes these models. As discussed in section III-D, practitioners should expect similar (not identical) ratings of quality for similar code. Thus, the mapping of different calculation approaches of quality to an ordinal letter rating needs calibration.

We have made the following contributions: i) an operationalization of the Quamoco quality model as a plugin to the SonarQube™ ii) a comparison of SQALE and Quamoco iii) a comparison of reported quality for 18 open source and 2 commercial projects; where quality was rated in the security, reliability and maintainability dimensions; and iv) an integrated visual dashboard for the Quamoco quality model.

In future work we intend to move this research in the following directions. Initial work will be to expand the study along the axes of language, projects, and quality models in order to better understand the relationships between quality models. This would include additional state of the art quality models such as the SIG Maintainability model [36], ColumbusQM [37], the Design Quality Model [38], a recent model developed by Nakai et al. [39], and the Squale quality model [40]. It would also be interesting to explore the perspectives of practitioners and managers on the effects that these models have on decision makers. Combining the results of these approaches would improve the community's understanding of quality models and how and why specific ratings are produced.

Acknowledgements

This research is funded by the Construction Engineering Research Laboratories (CERL), the US Army Corps of Engi-

neers, and the Department of Defense through an intermediary partnership with TechLink.

REFERENCES

[1] 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." IEEE, Jun. 2012, pp. 1133–1142. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6227106

[2] "Home | TechLink," Feb. 2017. [Online]. Available: http://techlinkcenter.org/

[3] J.-L. Letouzey and T. Coq, "The SQALE Analysis Model: An Analysis Model Compliant with the Representation Condition for Assessing the Quality of Software Source Code." IEEE, Aug. 2010, pp. 43–48. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5617180

[4] "SQALE | Software Quality Assessment based on Lifecycle Expectations," Feb. 2017. [Online]. Available: http://www.sqale.org

[5] "Construction Engineering Research Laboratory," Feb. 2017. [Online]. Available: http://www.erdc.usace.army.mil/Locations/CERL/

[6] "SonarQube | Continous Code Quality," Feb. 2017. [Online]. Available: http://www.sonarqube.org

[7] "ISO/IEC 9126-1:2001 Software Engineering – Product Quality – Part 1: Quality Model," Jun. 2001.

[8] "ISO/IEC 25010:2011 Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – Systm and Software Quality Models," Mar. 2011.

[9] S. Wagner, *Software Product Quality Control*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. [Online]. Available: http://link.springer.com/10.1007/978-3-642-38571-1

[10] R. Ferenc, P. Hegeds, and T. Gyimthy, "Software Product Quality Models," in *Evolving Software Systems*, T. Mens, A. Serebrenik, and A. Cleve, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 65–100, dOI: 10.1007/978-3-642-45398-4_3. [Online]. Available: http://link.springer.com/10.1007/978-3-642-45398-4_3

[11] 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.

[12] J. de Groot, A. Nugroho, T. Back, and J. Visser, "What is the value of your software?" in *Managing Technical Debt (MTD), 2012 Third International Workshop on*, Jun. 2012, pp. 37–44.

[13] D. Falessi and A. Voegele, "Validating and Prioritizing Quality Rules for Managing Techncial Debt: An Industrial Case Study," *MTD 2015-UNDER REVISION*, 2015.

[14] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, ser. FoSER '10. Santa Fe, New Mexico, USA: ACM, 2010, pp. 47–52. [Online]. Available: http://doi.acm.org/10.1145/1882362.1882373

[15] 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*, ser. MTD '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/1985362.1985364

[16] N. S. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spinola, "Towards an Ontology of Terms on Technical Debt." IEEE, Sep. 2014, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6974882

[17] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," *Software, IEEE*, vol. 29, no. 6, pp. 18–21, Dec. 2012.

[18] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," 2016. [Online]. Available: https://doi.org/10.4230/DagRep.6.4.110

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

[20] I. Griffith, D. Reimanis, C. Izurieta, Z. Codabux, A. Deo, and B. Williams, "The Correspondence Between Software Quality Models and Technical Debt Estimation Approaches." IEEE, Sep. 2014, pp. 19–26. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6974885

[21] R. K. Yin, *Case study research: design and methods*, 4th ed., ser. Applied social research methods. Los Angeles, Calif: Sage Publications, 2009, no. v. 5.

[22] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*, ser. Prentice Hall object-oriented series. Englewood Cliffs, NJ: PTR Prentice Hall, 1994.

[23] L. C. Briand, J. Wst, J. W. Daly, and D. Victor Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *Journal of Systems and Software*, vol. 51, no. 3, pp. 245–273, May 2000. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0164121299001028

[24] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design." ACM Press, 1991, pp. 197–211. [Online]. Available: http://portal.acm.org/citation.cfm?doid=117954.117970

[25] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[26] 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. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/016412129390077B

[27] M. Genero, J. Olivas, M. Piattini, and F. Romero, "Using Metrics to Predict OO Information Systems Maintainability," in *Advanced Information Systems Engineering*, G. Goos, J. Hartmanis, J. van Leeuwen, K. R. Dittrich, A. Geppert, and M. C. Norrie, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, vol. 2068, pp. 388–401, dOI: 10.1007/3-540-45341-5_26. [Online]. Available: http://link.springer.com/10.1007/3-540-45341-5_26

[28] S. Bangdiwala, "A graphical test for observer agreement," in *45th International Statistical Institute Meeting*, 1985, pp. 307–308.

[29] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, Apr. 1960. [Online]. Available: http://journals.sagepub.com/doi/10.1177/001316446002000104

[30] L. A. Goodman and W. H. Kruskal, "Measures of Association for Cross Classifications," *Journal of the American Statistical Association*, vol. 49, no. 268, p. 732, Dec. 1954. [Online]. Available: http://www.jstor.org/stable/2281536?origin=crossref

[31] M. G. Kendall, "A New Measure of Rank Correlation," *Biometrika*, vol. 30, no. 1/2, p. 81, Jun. 1938. [Online]. Available: http://www.jstor.org/stable/2332226?origin=crossref

[32] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945. [Online]. Available: http://www.jstor.org/stable/3001968

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

[34] D. Campbell and J. Stanley, *Experimental and Quasi-experimental Designs for Research*. Rand-McNally, 1963.

[35] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. [Online]. Available: http://link.springer.com/10.1007/978-3-642-29044-2

[36] I. Heitlager, T. Kuipers, and J. Visser, "A Practical Model for Measuring Maintainability." IEEE, Sep. 2007, pp. 30–39. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4335232

[37] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimothy, "A probabilistic software quality model." IEEE, Sep. 2011, pp. 243–252. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6080791

[38] R. Plsch, J. Bruer, C. Krner, and M. Saft, "Measuring, Assessing and Improving Software Quality based on Object-Oriented Design Principles," *Open Computer Science*, vol. 6, no. 1, Jan. 2016. [Online]. Available: http://www.degruyter.com/view/j/comp.2016.6.issue-1/comp-2016-0016/comp-2016-0016.xml

[39] H. Nakai, N. Tsuda, K. Honda, H. Washizaki, and Y. Fukazawa, "A SQuaRE-based software quality evaluation framework and its case study." IEEE, Nov. 2016, pp. 3704–3707. [Online]. Available: http://ieeexplore.ieee.org/document/7848750/

[40] 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." IEEE, Sep. 2009, pp. 531–534. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5306381