Synergies and conflicts among software quality attributes and bug fixes

Article *in* International Journal of Information Systems and Change Management · January 2017

DOI: 10.1504/IJISCM.2017.10007287

CITATION
1

READS
52

1 author:

Raed Shatnawi
Jordan University of Science and Technology
34 PUBLICATIONS **661** CITATIONS

SEE PROFILE

Synergies and conflicts among software quality attributes and bug fixes

Raed Shatnawi

Software Engineering Department,
Jordan University of Science and Technology,
Irbid, 22110, Jordan
Email: raedamin@just.edu.jo
and
College of Computer and Information Systems,
Al Yammamah University,
Riyadh, Kingdom of Saudi Arabia
Email: r_shatnawi@yu.edu.sa

Abstract: Quality has great effect on acceptance and behaviour of software products under operation. Quality has many perspectives for users, engineers and managers. These perspectives are represented formally in a quality model that is composed of both external and internal measures. External quality attributes can be measured indirectly using internal properties of the software product. Many external attributes are of interest to manager of software production and are combined in a framework or a quality model. However, these attributes may have interrelationships, positive or negative (conflicts). In addition, external quality attributes may have relationships with bugs in a component or system. In this research, we aim to find empirical evidence of the interrelationships among six quality attributes (reusability, flexibility, understandability, functionality, extendibility and effectiveness) and with bugs. We also demonstrate the evolution of these quality attributes in consecutive releases. Eleven open-source systems and their evolutions are considered to study the research objectives. The results could not show a relationship between the quality attributes and bugs in systems. The evolution of systems is consistent with the evolution in quality, i.e., quality increases in consecutive releases. There are some positive and low negative relationships among the quality attributes.

Keywords: software quality attributes; quality model for object-oriented design; QMOOD; bugs; faulty classes.

Reference to this paper should be made as follows: Shatnawi, R. (2017) 'Synergies and conflicts among software quality attributes and bug fixes', *Int. J. Information Systems and Change Management*, Vol. 9, No. 1, pp.3–21.

Biographical notes: Raed Shatnawi earned his PhD in Computer Science and MSc in Software Engineering from the University of Alabama in Huntsville. He is an Associate Professor in the Software Engineering Department in Jordan University of Science and Technology. His main interests are software metrics, software maintenance, and open source systems development.

1 Introduction

Product quality is measured by quality attributes and defects present in the product. Quality attributes predetermines intended behaviour of systems and as a result of bugs systems expected behaviour deviates, leading to poor quality of product (Shubhamangala et al., 2013). Software quality is measured from different perspectives, internal and external, to provide a comprehensive quality assessment. Many software quality models have been proposed to measure quality for different external characteristics, attributes or factors. These external attributes are more interesting to customers and developers of software, while internal attributes are more interesting for developers. External quality attributes are mapped to properties of software that can be measured using internal metrics. Software quality models provide a framework to measure both external and internal quality attributes. Internal quality can be measured using software metrics such as the well-known Chidamber and Kemerer (1994) metrics suite or any other comparable metrics.

Many quality models have proposed links between internal software properties and external quality attributes/factors. McCall et al. (1977) proposed a quality model as a hierarchy of factors, criteria, and measures. McCall et al. (1977) proposed 11 external quality factors: maintainability, flexibility, testability, portability, reusability, interoperability correctness (functionality), reliability efficiency, integrity and usability. Boehm et al. (1978) proposed a quality model that consists of three levels of characteristics: high-level, intermediate-level and primitive-level. It introduces seven external quality factors for three high-level characteristics: portability, as-is utilities (reliability, efficiency and usability), and maintainability (testability, understandability and modifiability). Each of these external attributes are mapped to many primitive sub characteristics (measures). FURPS/FURPS+ is another model similar to McCall and Bohem models (Grady, 1992). FURPS+ is an extended version by rational software (Kruchten, 2000). Dromey's quality model is another model similar to the McCall's, Boehm's and the FURPS (+) quality models (Dromey, 1995). Dromey's model focuses on the relationship between the attributes and the sub-attributes. It provides a mapping of software properties to software quality attributes. ISO/IEC 9126[1] (1991) is a standard model that defines quality as a set of product characteristics at two levels: internal and external. ISO model includes six quality attributes: functionality, reliability, efficiency, maintainability, portability and usability. These quality attributes are refined into more sub-attributes of software product. Squid (Bøegh et al., 1999) is a variant of the ISO 9126 model, but it depends on having a past-experience database to establish a relationship between internal and external quality characteristics. Bansiya and Davis (2002) proposed a hierarchal quality model for object-oriented design (QMOOD) derived from ISO model.

All these models are hierarchal and can be used to define measures and the relationships between the internal and external quality characteristics. However, they differ at the low level of the hierarchy and the measure definition. QMOOD model can be used at both the system and component levels and it is easy to assess, because it provides a quantitative assessment of external quality factors from the measurements of the internal design properties. QMOOD is a hierarchal quality model that is used to assess

external quality factors using design measures and can be applied to early stages (requirements and design) to ensure that software products have favourable internal properties. This model gives developers an opportunity to fix problems, remove irregularities and non-conformance to standards, and eliminate unwanted complexity early in the development life-cycle (Bansiya and Davis, 2002).

Software quality can be quantified by software measures as Bansiya and Davis have suggested in 2002. Alshayeb (2009) found a relationship between eight refactorings with five quality factors: adaptability, maintainability, understandability, reusability and testability. The results showed negative relationships between the applied refactorings and the software quality factors (adaptability, maintainability, understandability, reusability and testability). Rech (2009) proposed a quality framework for defect diagnosis. The application of the framework observed improvement on maintainability and productivity (finding defects). There are many research studies that used QMOOD to assess software quality. For example, O'Keeffe and O'Cinneide (2008) used QMOOD to evaluate alternative designs that can produce a better quality. In another study, Hsueh et al. (2008) used QMOOD to validate whether a design pattern is well-designed, i.e., it answers the question whether a proposed structural model really resolves the quality problems described in the intent of design patterns. Shatnawi and Li (2011) used QMOOD to assess the refactoring effect on software quality attributes, extendibility, effectiveness, flexibility and reusability. Ampatzoglou et al. (2011) evaluated the reusability of classes (as defined in QMOOD) that participate in design patterns and compared the reusability of each of these classes with the reusability of the pattern and the package. Khomh and Guéhéneuc (2008) also studied the impact of design patterns on quality attributes and found that patterns have a positive impact on quality attributes.

Furthermore, using QMOOD has many advantages: it can be used to measure software artefacts at many phases; it evaluates many external quality factors; it provides an assessment of the quality of the software artefact after evolution, which is consistent with the measurements of the external quality factors mentioned above; it gives one measurement for each quality factor, therefore making the comparison between releases easy to assess. Finally, QMOOD is insensitive to the design size (number of classes), because we compare normalised system measurements. Moses (2009) argued to measure software quality in a controlled way and that direct measurement of quality attributes should be encouraged to establish consistency.

In this research, we study the correlation between software quality attributes as defined in QMOOD and bugs at two levels, at class level and system level. This objective aims to find any correlation of the most critical quality attribute (bugs) and most desirable quality attributes in most systems. We also study the quality of evolving open-source systems using the QMOOD model. Measuring software quality requires data collection of many well-known software systems. We use 11 open-source systems and their evolution (three or more releases of each).

In the rest of this paper, we discuss the following. In Section 2, we provide evidence of related work on interrelationships among quality attributes. In Section 3, we provide a comprehensive explanation of research questions, QMOOD, metrics gathered and process of collection of metrics and fault data. In Section 4, analytical and statistical descriptions of the results are provided. We conclude our results in Section 5.

2 Related work

Software quality models also provide a framework of interrelationships between quality attributes. The relationship between a pair of quality attributes can be either positive, negative or neutral (no relation). A positive relationship between two attributes means if one quality attributes increases another attribute increases as well. A negative relationship means if one quality attributes increase another attribute is affected negatively and decreases. Some attribute pairs may not have any relationships and it means they are independent/neutral.

Studying the interrelationship is very important and have received attention in many previous research. Attributes interrelationships has effect on quality prioritisation starting from requirements (i.e., non-functional requirements), design (architecture evaluation), and code quality. Software product designers need to analyse trade-offs between conflicting quality attributes using quantitative evaluation to arrive at a better overall system (Barbacci et al., 1995).

McCall (1994) reported many relationships among quality attributes. Most relationships were either neutral or positive, and only two conflicts (flexibility with reliability and reusability with reliability). Reliability had positive relationship with usability and testability. Boehm and In (1996) proposed a tool (QARCC) to examine tradeoffs in quality attributes involved in software architecture. Using the tool, the authors found many conflicts such as cost/schedule with assurance, performance and reusability; and conflicts of interoperability and evolvability with performance. Kotonya and Sommerville (1997) have found a negative relationship between performance and security. These quality attributes were considered as non-functional requirements. Bosch (2000) also reported negative relationships among many quality attributes such as flexibility with runtime, and reusability with performance. Henningsson and Wohlin (2002) studied the relationship between quality attributes as proposed in McCall (1994) by conducting two surveys: research literature survey and survey with people from industry. They found, in qualitative terms, that quality attributes are dependent. However, no quantitative relations have been found. They used online survey to identify the relationships among the quality attributes. Al-Daajeh et al. (2012) introduce a framework to explore the nature of the relationships between different quality attributes at a low level with the aid of architectural strategies (i.e., tactics). García-Mireles et al. (2013) assessed quality of both code and design models in order to determine three quality attributes of software products in industrial settings: maintainability, security and usability. The authors also proposed a process framework that would manage the interactions between these quality requirements to manage conflicts between quality attributes that arise in a project. In a recent research, García-Mireles et al. (2015) studied the effect of traditional process reference models, such as CMM, CMMI or ISO 9001, on software equality attributes such as security, usability and reliability. García-Mireles et al. (2015) found increase in quality attributes in many empirical papers as effect of using these process improvement models, these principally being maintainability and reliability.

Software quality attributes are related to both software quality requirements and quality in architecture in most previous research. However, prioritising requirements and in particular quality attributes is not an easy task because of the innate relationships between these attributes (Dabbagh and Lee, 2013). The results of the previous research have shown differences in the interrelationships among quality attributes. The reasons for

these variations could be related to differences in definitions of the quality attributes, domains of use, and the experiences of the people who has constructed the comparisons (Berander et al., 2005). A problem with such relations is that in some cases it is not easy to determine the source of the definition of the quality attribute. The settings of the relationship is not defined, so it is hard to know the granularity of the relations as well and it may be described as generic relations that is applicable for a majority of projects (Henningsson and Wohlin, 2002). In addition, little empirical support were provided when software quality trade-offs are involved (Barney et al., 2012; García-Mireles et al., 2013).

3 Research methodology

This research aims to evaluate the correlation between bug fixes and external quality attributes as defined in a hierarchal quality model that is proposed in Bansiya and Davis (2002). This research provides more insights to the quality of well designed and developed systems in the open-source field. The results of this work help the developers in evaluating the quality of software. We evaluate software quality using six quality attributes: functionality, reusability, extendibility, understandability and effectiveness. In addition, we study the bug fixes in classes and the correlation with other quality attributes. Software quality models provide a framework to evaluate quality from different perspectives. A quality model provides measurement scales of software properties and attributes to provide indicators of software quality. In addition, these measurements are used to compare different versions of a software as well as quality of different software. In this research, we aim to answer the following questions:

- RQ1: are there correlations between software bug fixes and quality attributes as defined in QMOOD in each system at class level?

  The quality attributes are measured for each class and then correlations are assessed between each quality attribute from one side and with bugs fixes in a class.

- RQ2: Do quality attributes provide evidence of software evolution when used to evaluate consecutive releases of a software?

  We measure the quality attributes for many releases of each software and evolution of these attributes is discussed in details.

- RQ3: are there correlations between number of bugs in a software and quality attributes as defined in QMOOD at system level?

  The quality attributes are measured for each system by summing values of each attribute for all classes/modules. The values are provided in one table and a correlation is conducted with number of total bugs fixed in each system. The statistical significance of each correlation is also reported.

To answer these questions, we use 11 open-source systems and their evolution data. In the following, we provide details of quality attributes as defined in QMOOD. We also present how software were analysed and how data were collected.

3.1 The QMOOD

To achieve the objectives of this research, we need a model to assess software quality. The model measures both the internal and external quality characteristics. Internal characteristics are used to measure software properties such as design size, hierarchies, abstraction, encapsulation, coupling, cohesion, composition, inheritance, polymorphism, messaging and complexity. QMOOD assesses six external quality attributes, 11 internal design properties, and OO design components such as attributes, methods, classes, composition and inheritance hierarchies.

The definitions of the quality attributes are provided as follows:

- Reusability defines how a pre-defined component can be reuse in a new problem in object-oriented design and implementation with low effort.
- Extendibility allows the incorporation of new requirements in design and implementation with existing properties.
- Understandability measures the ability and easiness of learning the design and implementation, and it measures the degree of complexity.
- Functionality defines the responsibilities of classes in a design.
- Extendibility: measures the ability of an existing design to incorporate new requirements.
- Effectiveness measures the ability of a design to achieve required functionality and behaviour in object-oriented paradigm.

QMOOD links these external attributes to many design properties that are quantified by software measures. The software properties that are used in this study are defined in Table 1. Each design property can be measured using already proposed metrics in literature. As suggested by Bansiya and Davis (2002), other metrics can be suggested from literature. In this research, metrics of Chidamber and Kemerer (1994) and Bansiya and Davis (2002) are used to assess quality attributes. The metrics data were collected using CKjm tool and provided by the promise data repository (Jureczko and Madeyski, 2010; Jureczko and Spinellis, 2010). Some metrics should be measured at system level such as design size and number of hierarchies. Design size is set to one at low level measurement for a class or module. The number of hierarchies are counted using two measures, depth of inheritance (DIT) and number of children (NOC). A hierarchy exists if the DIT value is one (root class) and the module has one or more children, otherwise zero is assigned. The number of hierarchies in a system is then the summation of the hierarchy metric of all classes. Each quality attribute is a function of many software properties (measures) as shown in Table 2.

Bansiya and Davis (2002) have proposed and validated quality indexes empirically as shown in Table 2. Proportional weights were assigned to indicate the influence of the properties on a quality factor. Bansiya and Davis (2002) conducted an empirical study to determine the weights and the model was validated on the evolution of two real systems. The weights (+0.5, +1) were used initially if a measure positively influences a quality attribute and the weights (–0.5, –1) were used for negative influences. These weights were changed to ensure that the sum of weighted values added to +1; for example, the sums of weighted values of the properties used to measure reusability add to 1 (Bansiya

1997). An increase on a quality attribute after maintenance indicates an improvement of quality; otherwise a decrease indicates a quality deterioration. QMOOD was selected for this work because it provides quality indexes that are easy to interpret using software measures and can be calculated for individual classes. QMOOD provides these measures and how to assess them. Metrics in Table 1 are used to calculate the quality indexes. These metrics have different scales, some metrics are measured as a ratio while others as either interval or absolute values. We use min-max normalisation to reduce the effect of some metrics by transformation metric values to be in the range (0, 1). Min-max normalisation preserves the relationships among the original data values (Han et al., 2011). Min-max is a linear transformation that can be calculated as follows:

$$v' = (v - \min)/(\max - \min)$$

Table 1 OO metrics to measure OO design and implementation properties

Property	*QMOOD metric*	*Definition*	*CKjm metric*
Design size in classes	The number of classes at the measurement level	This metric is fixed to 1 for all classes. The number of classes is the design size for the whole system.	DSC (Bansiya and Davis, 2002)
Hierarchies		This metric is measured either 1 if the class is root class with children	DIT = 1 and NOC > = 1
Abstraction	Average number of ancestors (ANA)	The average number of classes from which a class inherits information. It is determined by class inheritance structure in design by computing the number of classes along all paths from the root class to other classes in the inheritance structure. In implementation, depth of inheritance tree (DIT) metric is equivalent to ANA metric in design.	DIT (Chidamber and Kemerer, 1994)
Encapsulation	Data access metric (DAM)	The ratio of the number of private and protected attributes to the total number of attributes declared in the class. Range is 0 to 1, and high values of DAM are desired.	DAM (Bansiya and Davis, 2002)
Coupling	Direct class coupling (DCC)	Count the different number of classes that a class is directly related to. It is determined through attributes declaration and message passing in methods.	CBO (Chidamber and Kemerer, 1994)
Cohesion	Cohesion among methods of class (CAM)	Computes the relatedness among methods of a class based upon the parameter list of the methods. The metric is computed using the summation of the intersections of parameters of a method with the maximum independent set of all parameter types in the class. A metric closed to 1 is preferred (range 0 to1).	CAM (Bansiya and Davis, 2002)
Composition	Measure of aggregation (MOA)	Counts the number of data declaration whose types are user defined classes, and it is realised by using attribute declaration.	MOA (Bansiya and Davis, 2002)

Table 1 OO metrics to measure OO design and implementation properties (continued)

Property	*QMOOD metric*	*Definition*	*CKjm metric*
Inheritance	Measure of functional abstraction (MFA)	The ratio of the number of methods inherited by a class to the total number of methods of the class (range 0 to 1).	MFA (Bansiya and Davis, 2002)
Polymorphism	Number of polymorphic methods (NOP)	This metric is a count of the methods that can exhibit polymorphic behaviour, and such methods in C++ are marked as virtual. In Java all public and protected methods can be overridden and so polymorphic.	NPM (Bansiya and Davis, 2002)
Messaging	Messaging (class interface size)	The NPM metric simply counts all the methods in a class that are declared as public. The metric is known also as class interface size (CIS)	NPM (Bansiya and Davis, 2002)
Complexity	Number of methods (NOM)	The number of all methods defined in a class. It is equivalent to WMC (Chidamber and Kemerer, 1994).	WMC (Chidamber and Kemerer, 1994)

Table 2 Quality indices of external quality attributes as adapted from Bansiya and Davis (2002)

Quality attribute	*Quality index 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
Understandability	–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

3.2 Data sources

The research methodology is validated on multiple-releases of 11 open-source systems. The source code for all systems are available online as provided in Table 3. These systems are developed in Java and represent several domains. We have used the collected metrics data for many releases; the number of releases and the number of classes in last release are shown in the last two columns. The metrics data were collected using an open-source tool, Ckjm (http://www.spinellis.gr/sw/ckjm/) and reported by the promise data repository by Jureczko and Madeyski (2010) and Jureczko and Spinellis (2010). The fault data for the systems under investigation were collected from repositories of the projects (Jureczko and Madeyski, 2010; Jureczko and Spinellis, 2010). The authors have used BugInfo (https://kenai.com/nonav/projects/buginfo) used to collect the fault data. BugInfo searches software versioning by studying the code repositories (subversion or

CVS). If a log description includes a fix of a particular bug, then all affected classes are extracted using regular expressions from bug description and marked as faulty.[2] When a bug is found in a log description, then faults count is incremented. The faults distributions are shown for each system in Table 3.

Table 3 Systems under investigation

System	*Brief description*	*#Releases*	*#Classes in the last release*
Ant	A Java building tool http://ant.apache.org	5	745
Camel	A versatile open-source integration framework http://camel.apache.org	4	965
Ivy	A dependency manager focusing on flexibility and simplicity. http://ant.apache.org/ivy/	3	352
Jedit	A Java IDE and editor http://jedit.org/	5	492
Log4J	A logging services http://logging.apache.org/log4j	3	205
Lucene	develops open-source search software http://lucene.apache.org/	3	340
POI	Java API for Microsoft documents http://poi.apache.org/	4	442
Synapse	Enterprise service bus (ESB) http://synapse.apache.org/	3	256
Velocity	A powerful Java-based template engine that renders data from plain Java objects to various formats	3	229

3.3 Pearson correlation

We use Pearson correlation to find the correlation between two variables (quality attributes and faults or bugs in software). Pearson correlation assumes that the variables under consideration were measured on ratio/interval scale. The correlation coefficient is between –1 and 1 and the value is interpreted using the rule of thumbs provided in Table 4 (Hinkle et al., 2003). If one variable tends to increase as the other decreases, the correlation coefficient is negative. Conversely, if the two variables tend to increase together the correlation coefficient is positive. For a two-tailed test of the correlation:

$H_0 : \rho = 0$, versus

$Hi : \rho \neq 0$

where ρ is the correlation between a pair of variables.

Table 4 Interpretation of a correlation coefficient

Correlation	*Interpretation*
.70 to 1.0 (–.70 to –1.0)	High positive (negative) correlation
.50 to .70 (–.50 to –.70)	Moderate positive (negative) correlation
.30 to .50 (–.30 to –.50)	Low positive (negative) correlation
.00 to .30 (.00 to –.30)	Negligible correlation

This test is used to answer RQ3 to find the statistical significance of the correlation of quality attributes with number of faulty classes and with the number of fixed bugs in a system.

4 Results analysis

To answer the three questions, we analysed 11 open-source systems and their evolution. We calculated quality indexes using the formulas in Table 2 for each system.

4.1 Research question 1: quality attribute versus faulty/number of bugs

We conducted the correlation analysis between the six quality attributes and the number of bug fixed in each class. The correlation coefficients are shown in Table 5. The results provide no evidence of strong correlation between quality attributes and number of bugs in classes. The correlation is negligible (< |0.30|) for most systems and few have low correlation. Therefore, we conclude that there is no relationship between suggested quality attributes and number of bugs at the class level. We repeated the correlation analysis between the six quality attributes and the faulty/not faulty classes. The correlation coefficients are shown in Table 6. The results provide no evidence of strong correlation between quality attributes and faulty classes. The correlation is either has negligible (< |0.30|) or low correlation with faulty classes. Therefore, we conclude that there is no relationship between suggested quality attributes and faulty/not faulty classes. These results show that the QMOOD quality attributes measure a different dimension on quality that is not measured by number of bugs or faulty classes.

We also conducted the correlation among the quality attributes for each system (last release only). The results of correlations are summarised for brevity in Table 7. The results of correlation analysis are conducted at the class level for each system. We observe few synergies/conflicts between quality attributes. We find a positive correlation between effectiveness and extendibility. Effectiveness refers to a design's ability to achieve a desired functionality and behaviour using object-oriented design concepts and techniques while extendibility refers to the presence of properties in existing design that allows the addition of new functionality and behaviour. Similarly, we notice a moderate correlation between effectiveness and flexibility. Flexibility in design allows the incorporation of new functionality while effectiveness refers to design's ability to achieve required functionality. Functionality and extendibility have weak correlations in eight systems. Again, systems functionality is an indicator of more extension of the software for different purposes. We observe negative correlations between understandability and functionality. This conflict comes from the fact that systems that provide more responsibilities are more difficult to understand and maintain. We also observe negative correlations between understandability and extendibility. This conflict comes from the fact that systems that have properties of extension are more difficult to understand and maintain. Reusability shows four negative correlations with flexibility. Reusability measures the ability of design to be reused in other contexts, while flexibility provides a design with ability to adapt to new functionality.

Table 5 Correlation coefficients of number of bugs versus quality attributes

Bugs	*Reusability*	*Flexibility*	*Understandability*	*Functionality*	*Extendibility*	*Effectiveness*
Ant 1.7	–0.05	0.35	–0.27	0.18	0.03	0.18
Camel 1.6	–0.16	0.15	–0.18	0.00	–0.05	0.05
Ivy 2.0	–0.21	0.25	–0.18	0.15	–0.06	0.12
Jedit 4.3	0.00	–0.04	–0.07	0.08	0.05	0.01
Log4J 1.2	0.02	0.27	–0.16	0.42	0.11	0.26
Lucene 2.4	0.03	0.44	–0.31	0.43	0.07	0.32
POI 3.0	0.13	0.32	–0.21	0.13	–0.07	0.13
Synapse 1.2	–0.18	0.22	–0.12	0.11	–0.09	0.17
Velocity 1.6	–0.07	0.28	–0.23	0.07	–0.08	0.11
Xalan 2.7	–0.01	–0.03	–0.02	–0.05	–0.01	–0.02
Xerces 1.4	0.00	0.02	0.01	0.00	0.01	0.01

Table 6 Correlation coefficients of faulty/not faulty versus quality attributes

Faulty	*Reusability*	*Flexibility*	*Understandability*	*Functionality*	*Extendibility*	*Effectiveness*
Ant 1.7	–0.11	0.34	–0.18	0.18	0.05	0.19
Camel 1.6	–0.09	0.13	–0.07	–0.05	0.00	0.07
Ivy 2.0	–0.21	0.22	–0.15	0.12	–0.05	0.12
Jedit 4.3	–0.01	–0.03	–0.07	0.10	0.05	0.01
Log4J 1.2	–0.01	0.14	0.03	0.12	–0.06	0.06
Lucene 2.4	–0.09	0.28	–0.06	0.20	0.06	0.26
POI 3.0	–0.06	0.33	–0.07	0.38	0.02	0.24
Synapse 1.2	–0.21	0.25	–0.10	0.25	0.02	0.28
Velocity 1.6	–0.06	0.25	–0.20	0.11	–0.11	0.06
Xalan 2.7	0.05	0.02	0.04	–0.05	0.03	0.02
Xerces 1.4	0.03	0.06	0.10	–0.02	–0.03	0.00

Table 7 Summary of correlations among six quality attributes

	Reusability	*Flexibility*	*Understandability*	*Functionality*	*Extendibility*
Flexibility	4 low negative				
Understandability		1 low positive			
Functionality		1 moderate positive 5 low positive	9 low negative		
Extendibility			2 moderate negative 6 low negative	8 low positive	
Effectiveness	1 low negative	2 high positive 9 moderate positive	1 low negative	4 moderate positive 6 low positive	1 moderate positive 10 high positive

4.2 Research question 2: statistical significance of correlation between quality attributes and faulty/number of bugs at the system level

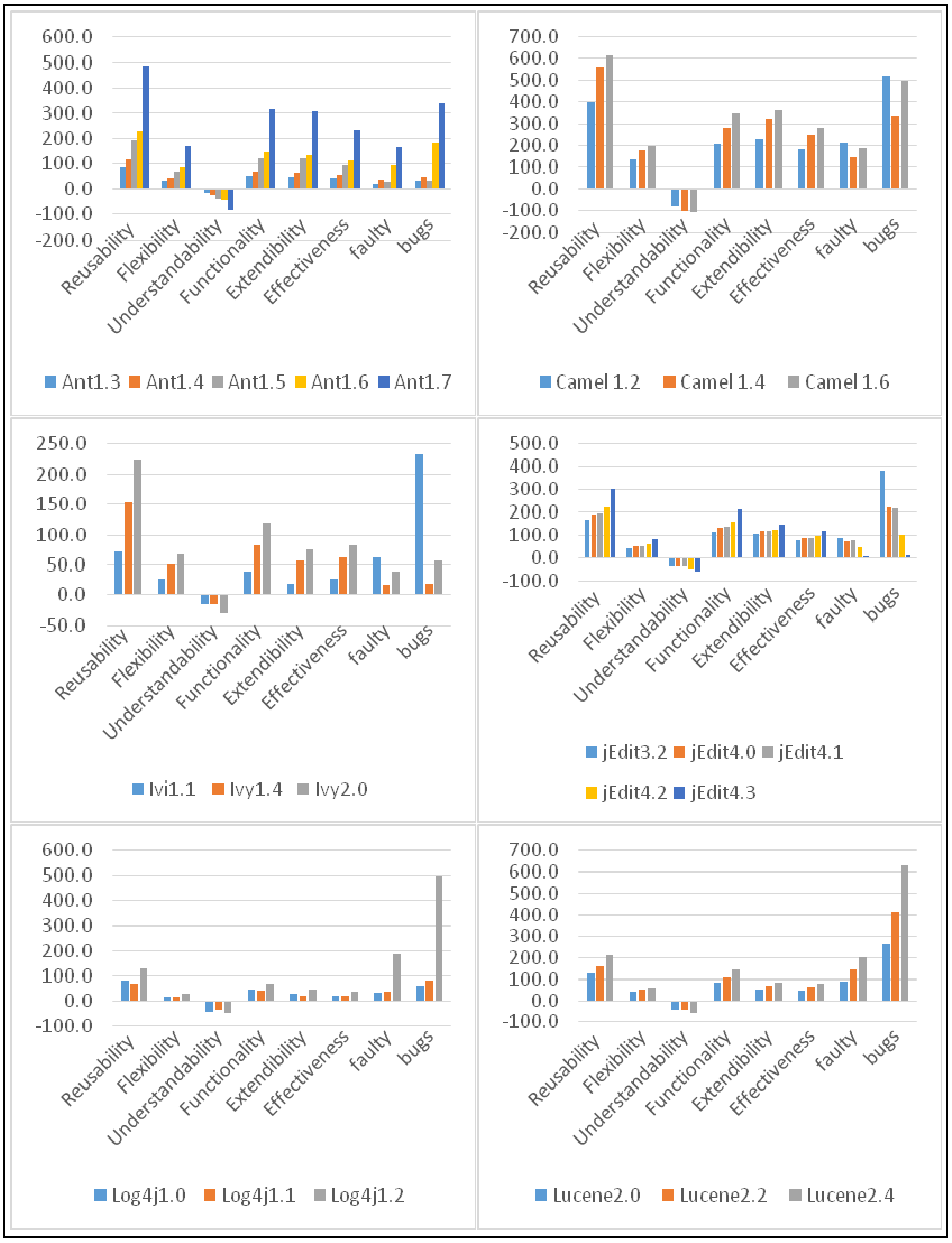
In this question, we use a correlation analysis to find the correlations between each quality attribute and faulty classes and number of bugs in each system. The correlation coefficient and statistical significance is shown in Table 8. There are many significant correlations as shown in Table 8 between faulty classes in a system and four quality factors (reusability, understandability, functionality and extendibility). These number of faulty classes increases for increase in these quality attributes. Understandability is measured in negative values and this explains why we have negative correlation with faulty classes. On the other hand, number of bugs in a software has significant correlations with only two quality attributes, understandability and functionality. We also notice the correlation is moderate with functionality. These results show whenever there is an increase in reusability, extendibility and functionality of systems, bugs are increased as well. Bugs increase in a system whenever understandability decreases.

Table 8 Correlation coefficients for quality attributes with faulty/bugs at system level

		Reusability	*Flexibility*	*Understandability*
Faulty	r	*0.564*	0.238	*–0.839*
	p-value	*0.071**	0.48	*0.001***
Bugs	r	0.458	0.139	*–0.896*
	p-value	0.156	0.684	*0***
		Functionality	*Extendibility*	*Effectiveness*
Faulty	r	*0.785*	*0.554*	0.432
	p-value	*0.004***	*0.077**	0.185
Bugs	r	*0.577*	0.409	0.28
	p-value	*0.063**	0.212	0.405

Notes: *Significant at 0.10 level. **Significant at 0.05 level.

Figure 1 (a) The evolution of six attributes in Ant, Camel, Ivy, JEdit, Log4j and Lucene (b) The evolution of six attributes in POI, Synapse, Velocity, Xalan, and Xerces (see online version for colours)



(a)

Figure 1 (a) The evolution of six attributes in Ant, Camel, Ivy, JEdit, Log4j and Lucene (b) The evolution of six attributes in POI, Synapse, Velocity, Xalan, and Xerces (continued) (see online version for colours)

(b)

4.3 Research question 3: evolution assessment

The quality attributes of many releases of the 11 systems are calculated as shown in Figure 1. The trend in five quality attributes (reusability, flexibility, functionality, extendibility and effectiveness) are the same, i.e., the values are increasing for consecutive releases. Software reusability follows this trend except in Log4J (Version 1.1), the reusability is decreasing in the second release. Flexibility follows this trend except in Log4J (Version 1.1) and Xalan (Version 2.7). Software functionality follows the same trend except in Log4J (Version 1.1), and POI (Version 3.0). Software extendibility follows the same trend except in Log4J (Version 1.1), while velocity has opposite trend (i.e., extendibility decreases in consecutive releases). The Understandability values are negative and indicate understandability is low for large systems. The values of understandability decreases for consecutive releases, and shows understandability reduces as systems evolve. Software understandability in Log4J and velocity does not follow this trend for all releases.

We also studied the trend in the number of faulty classes and number of bugs in evolution of systems. For five systems only (Ant, Log4J, Lucene, Synapse, and Xalan), there is a trend in number of faulty classes or number of bugs in consecutive releases. The number of faulty classes and number of bugs in these systems are increasing in later releases. Five systems out of 11 systems is not an evidence that that faulty classes or number of bugs follow the same trend as QMOOD quality attributes. In RQ1, we found no significant correlation between QMOOD quality attributes and faulty classes/number of bugs in a class.

5 Threats to validity

In this section, we discuss the validity threats of this study as follows:

- *Internal validity threats*: this research does not consider the impact of many development issues that are not usually revealed in the open-source systems including the software development process. However, all applications under study represent the open-source development. Many metric collection tools exist and some may use variant definitions for some metrics. For example, WMC has two main definitions. However, the quality model under consideration is flexible and various definitions of a metric can be applied similarly.
- *External validity threats*: The study considers only 11 applications in Java. The results of this research may not be generalised to other projects. However, the proposed methodology is easy to apply once the metrics data are collected. There are many other quality attributes that have been already defined and few of them were studied to find conflicts on synergies. However, the focus of this research is on the QMOOD model and its constituents only.

6 Conclusions

Software quality is measured by assessing many quality attributes and bugs present in the product. Software quality is measured from different perspectives for different purposes. Quality has many desirable and critical attributes. These attributes may have synergies or conflicts. It is necessary to know which pairs of quality attributes have conflicts to take in consideration during design and development phases. In this research, we study the interrelationships between six quality attributes and software bugs. The study is conducted at both the class and system level. The quality attributes are measured at the class and compared using the Pearson correlation. We could not find any interrelationships at the class level. We repeated the correlation analysis at the system level and we found some positive or negative relationships. These results raise the awareness of conflicts in some quality attributes. We also studied the evolution of quality for many releases of each system. We also studied the evolution of bug fixes in consecutive releases. The trends observed for the quality attributes are almost similar for most systems. The same trend can be observed for the number of bug fixes, but for only four systems. Number of bug fixes do not follow a clear trend or pattern as shown in the evolution of systems. The interrelationships that exist at the system levels could not be observed for classes. The quality attributes are better be measured for systems rather than for classes as shown in these results.

In the future, we aim to investigate more conflicts or synergies among other quality attributes not considered in this research. A mixture of both empirical and anecdotal analysis (e.g., surveying software engineers) may provide another dimension to the results.

References

Al-Daajeh, S.H., Al-Qutaish, R.E. and Al-Qirem, F. (2012) 'A tactic-based framework to evaluate the relationships between the software product quality attributes', *International Journal of Software Engineering*, Vol. 5, No. 1, pp.5–26.

Alshayeb, M. (2009) 'Empirical investigation of refactoring effect on software quality', *Information and Software Technology*, Vol. 51, No. 9, pp.1319–1326.

Ampatzoglou, A., Kritikos, A., Kakarontzas, G. and Stamelos, I. (2011) 'An empirical investigation on the reusability of design patterns and software packages', *Journal of Systems and Software*, Vol. 84, pp.2265–2283.

Bansiya, J. (1997) *A Hierarchical Model for Quality Assessment of Object-Oriented Designs*, PhD dissertation, Univ. of Alabama in Huntsville.

Bansiya, J. and Davis, C. (2002) 'A hierarchical model for object-oriented design quality assessment', *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, pp.4–17.

Barbacci, M.R., Klein, M.H., Longstaff, T.A. and Weinstoc, C.B. (1995) *Quality Attributes*, Technical Report, CMU/SEI-95-TR-021, Software Engineering Institute, Carnegie Mellon University, December.

Barney, S., Petersen, K., Svahnberg, M., Aurum, A. and Barney, H. (2012) 'Software quality trade-offs: a systematic map', *Information and Software Technology*, Vol. 54, No. 7, pp.651–662.

Berander, P., Damm, L., Eriksson, J., Gorschek, T., Henningsson, K., Jonsson, P., Kagstrom, S., Milicic, D., Martensson, F., Ronkko, K. and Tomaszewski, P. (2005) *Software Quality Attributes and Trade-offs*, Blekinge Institute of Technology.

Bøegh, J., Depanfilis, S., Kitchenham, B. and Pasquini, A. (1999) 'A method for software quality planning, control and evaluation', *IEEE Software, Researcher's Corner*, Vol. 16, No. 2, pp.69–77.

Boehm, B. and In, H. (1996) 'Identifying quality-requirement conflicts', *IEEE Software*, Vol. 13, No. 2, pp.25–35.

Boehm, B.W., Brown, H. and Lipow, M. (1978) *Quantitative Evaluation of Software Quality*, TRW Systems and Energy Group.

Bosch, J. (2000) *Design and use of Software Architecture – Adopting and Evolving a Product-Line Approach*, Addison-Wesley, Great Britain.

Chidamber, S.R. and Kemerer, C.F. (1994) 'A metrics suite for object oriented design', *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp.476–493.

Dabbagh, M. and Lee, S. (2013) 'A consistent approach for prioritizing system quality attributes', *14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp.317–322.

Dromey, R.G. (1995) 'A model for software product quality', *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, pp.146–163.

García-Mireles, G., Moraga, M., García, F. and Piattini, M. (2015) 'Approaches to promote product quality within software process improvement initiatives: a mapping study', *Journal of Systems and Software*, Vol. 103, No. 103, pp.150–166.

García-Mireles, G.A., Moraga, M.A., Garcia, F. and Piattini, M. (2013) 'A framework to support software quality trade-offs from a process-based perspective', in *EuroSPI*, pp.96–107.

Grady, R.B. (1992) *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall.

Han, J., Kamber, M. and Pei, J. (2011) *Data Mining: Concepts and Techniques*, 3rd ed., The Morgan Kaufmann Series in Data Management Systems.

Henningsson, K. and Wohlin, C. (2002) 'Understanding the relations between software quality attributes – a survey approach', *Proceedings 12th International Conference for Software Quality*, Proceedings on CD, Ottawa, Canada.

Hinkle, D.E., Wiersma, W. and Jurs, S.G. (2003) *Applied Statistics for the Behavioral Sciences*, 5th ed., Houghton Mifflin, Boston.

Hsueh, N., Chu, P. and Chu, W. (2008) 'A quantitative approach for evaluating the quality of design patterns', *Journal of Systems and Software*, Vol. 81, No. 8, pp.1430–1439.

ISO/IEC 9126 (1991) *Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use*, Int'l Organization for Standardization.

Jureczko, M. and Madeyski, L. (2010) 'Towards identifying software project clusters with regard to defect prediction', in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, pp.1–10.

Jureczko, M. and Spinellis, D. (2010) 'Using object-oriented design metrics to predict software defects', in *Proceedings of the 5th International Conference on Dependability of Computer Systems*, pp.69–81.

Khomh, F. and Guéhéneuc, Y. (2008) 'Do design patterns impact software quality positively? software maintenance and reengineering', *CSMR 2008, 12th European Conference on (CSMR'08)*, pp.274–278.

Kotonya, G. and Sommerville, I. (1997) *Requirements Engineering*, Wiley, West Sussex, England.

Kruchten, P. (2000) *The Rational Unified Process an Introduction*, 2nd ed., Addison Wesley Longman, Inc.

McCall, J.A. (1994) *Encyclopedia of Software Engineering*, Vol. 2, John Wiley & Sons, New York.

McCall, J.A. Richards, P.K. and Walters, G.F. (1977) *Factors in Software Quality*, Vols. 1, 2, and 3, Nat'l Tech. Information Service.

Moses, J. (2009) 'Should we try to measure software quality attributes directly?', *Software Qual. J.*, Vol. 17, No. 2, pp.203–213.

O'Keeffe, M. and O'Cinneide, M. (2008) 'Search-based refactoring for software maintenance', *Journal of Systems and Software*, Vol. 81, No. 4, pp.502–516.

Rech, J. (2009) *Context-sensitive Diagnosis of Quality Defects in Object-Oriented Software Systems*, PhD thesis, University of Hildesheim, Department IV, Hildesheim.

Shatnawi, R. and Li, W. (2011) 'An empirical assessment of refactoring impact on software quality using a hierarchical quality model', *International Journal of Software Engineering and Its Applications*, Vol. 5, No. 4, pp.127–150.

Shubhamangala, B.R., Suma, V., Amarendra Reddy, P. and Goodluck Singh, C. (2013) 'Quality attribute focused multilayer requirement elicitation: judicious approach to drive business value', in *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp.2069–2075.

Notes

1 ISO 9126 is software product quality standard that was superseded by SQuaRE series of International Standards (ISO/IEC 25000).

2 A class is marked faulty if there is at least one bug fix, otherwise the class is marked as not faulty.