Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 970 – 975

Procedia Computer Science

3rd International Workshop on Computational Antifragility and Antifragile Engineering (ANTIFRAGILE 2016)

Improving software applications quality by considering the contribution relationship among quality attributes

Fernando Pinciroli*

Faculty of Informatics and Design, Champagnat University, (5501) Godoy Cruz, Argentina

Abstract

There is a number of quality models for the evaluation of software products. The final objective for these models is to obtain a measure for the global quality of the software product and for each individual quality characteristic in order to select the more accurate solutions to improve the lasts and, through them, the whole software product. However, software quality attributes have positive and negative contribution relationships among them, which in fact restrict the selection of the best solutions. As one of the solutions to improve some of them could negatively influence another one, the global quality could have a small improvement, none improvement at all or what is worse, it could degrade. For this reason, we should measure the impact of the selected set of solutions in order to choose the set presenting the best results for improvement. With these metrics, it is possible to calculate the set of solutions that would improve a software product with the best quality gain according to the "system identity", composed by a set of quality attributes and its respective values, which therefore is one of the objectives of antifragility.

Peer-review under responsibility of the Conference Program Chairs

Keywords: Software quality; Software quality models; Software quality attributes; System/software product quality; System identity

1. Introduction

Form the quote attributed to Lord Kelvin: "If you cannot measure it, you cannot improve it", we can say that in order to improve software and web applications we should consider these three following actions: *to define*, *to*

* Corresponding author. Tel.: +54-261-424-8443; fax: +54-261-424-8443.
E-mail address: pincirolifernando@uch.edu.ar

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2016.04.194

measure and to improve. GOCAME[1], which stands for Goal-Oriented Context-Aware Measurement and Evaluation, is a strategy to measure and to evaluate the quality of different focuses –systems from their internal and external points of views, systems in use, resources and processes–, which allows reaching those three goals. The GOCAME strategy is made up of a conceptual base and a framework that explicitly and formally specifies the measurement and evaluation concepts, properties, relationships and constraints.

In Olsina et al.'s work[2], we can observe the quality attributes structure that was developed to measure the usability quality of Jira for its 1.0 and 1.1 versions using GOCAME. As it may be clearly seen on the tree-structure, it is used to categorize and to present the quality attributes, the qualifications obtained for each leaf of the tree, the accumulated qualifications for each attribute category on the nodes and, finally, the global improvement from the first version and the following one.

	JIRA v.1		JIRA v.1.1	
Characterisitcs and Attributes	EI	P/G I	EI	P/G I
1. Actual Usability		53.3%		67.0%
1.1. Effectiveness		73.2%		86.7%
1.1.1. *Sub-Task Correctness*	86.4%		91.9%	
1.1.2. *Sub-Task Completeness*	87.9%		95.5%	
1.1.3. *Task Successfulness*	45.5%		72.7%	
1.2. Efficiency		29.3%		42.8%
1.2.1. *Sub-Task Correctness Efficiency*	37.4%		44.3%	
1.2.2. *Sub-Task Completeness Efficiency*	37.5%		47.3%	
1.2.3. *Task Successfulness Efficiency*	13.1%		36.8%	
1.3. Learnability in use		57.3%		71.6%
1.3.1. *Sub-Task Correctness Learnability*	78.8%		75.1%	
1.3.2. *Sub-Task Completeness Learnability*	26.4%		77.3%	
1.3.3. *Task Successfulness Learnability*	66.7%		62.5%	

Fig. 1. Improvement of Jira's usability quality between two consecutive versions (taken from 2).

The measurement of quality attributes allows us to identify those attributes that could or should be improved in order to increase the global quality of a web or software application. In the previous example, the attributes 1.1.1, 1.1.2 and 1.1.3 could be enhanced, but the rest of the attributes should be improved.

A number of possible solutions can be considered to enhance the quality attributes, and certainly some of them were applied to the Jira's version 1.0 to obtain better global results for the version 1.1. However, if we look at some particular attributes, we can see that there was a descent on the qualifications. For instance, *"Sub-task correctness learnability"* and *"Task successfulness learnability"* got lower qualifications on the last version. Even though it was not possible to explain the exact cause of this degradation on some particular attributes, we can suppose that this situation was caused by the negative contributions that exist among quality attributes. This is the point that we want to focus on: to consider the contribution relationship among quality attributes before making a selection among different possible solutions.

2. Contribution relationships among software quality attributes

Quality attributes, also known as non-functional attributes, can relate each other positively or negatively when they are considered together on a software application. For instance, we can ask for portability on a software application because we want it to run on different mobile operating systems, but this portability will produce a negative impact on other quality attributes. For example, we could not reach all the performance needed due to portability, since we are not able to make the most with specific resources of each operating system. On the other hand, portability could be positively collaborating with other quality attributes like reusability.

Wiegers[3] explains that contributions among quality attributes can be represented by means of a matrix where a plus sign, a minus sign and a blank cell are respectively indicating the positive, negative and the indifferent relationships among them.

	Availability	Efficiency	Installability	Integrity	Interoperability	Modifiability	Performance	Portability	Reliability	Reusability	Robustness	Safety	Scalability	Security	Usability	Verifiability
Availability								+		+						
Efficiency	+				−	−	+	−			−		+		−	
Installability	+								+					+		
Integrity			−		−		−			−		+		+	−	−
Interoperability	+		−	−			−	+	+		+	−		−		
Modifiability	+		−				−		+	+			+			+
Performance		+			−	−		−			−		−		−	
Portability		−			+	−	−			+				−	−	+
Reliability	+	−		+		+	−				+	+		+	+	+
Reusability		−		−	+	+	−	+						−		+
Robustness	+	−	+	+	+		−		+			+	+	+	+	
Safety		−		+	+		−				+			+	−	−
Scalability	+	+		+			+	+	+		+					
Security	+			+	+		−	−	+		+	+			−	−
Usability		−	+				−	−	+		+	+				−
Verifiability	+		+	+		+			+	+	+	+		+	+	

Fig. 2. Positive and negative relationships among quality attributes (taken from 3).

In addition to this, if we fold the matrix along its main diagonal, we could find some cells with two plus signs, two minus signs or with both a minus and a plus sign. This is caused by the combined effect of contribution relationships among attributes.

3. Analyzing the impact of negative relationships among quality attributes

Providing an example, take as a consideration an automated teller machine where we are evaluating three quality attributes: usability, security and performance. Some of these attributes were subdivided into a new sublevel according to SQuaRE's system and software quality model for system/software product quality[4].

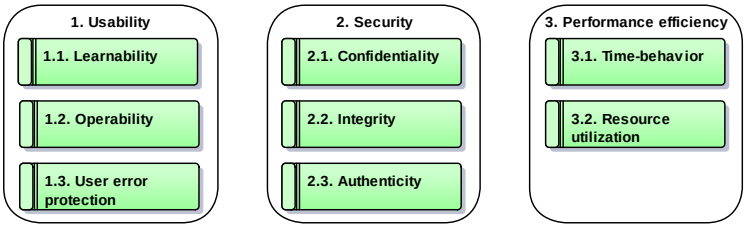

Fig. 3. SQuaRE's system and software quality model for system/software product quality.

Let us now suppose that the technique to measure the quality level for these attributes was already applied and it was found necessary to improve some of them: *integrity*, *authenticity* and *time-behavior*. In fact, there must be a number of alternatives as possible solutions to enhance these attributes, but each of them could have a different level of impact on the rest of the quality attributes, according to the contribution relationships among them.

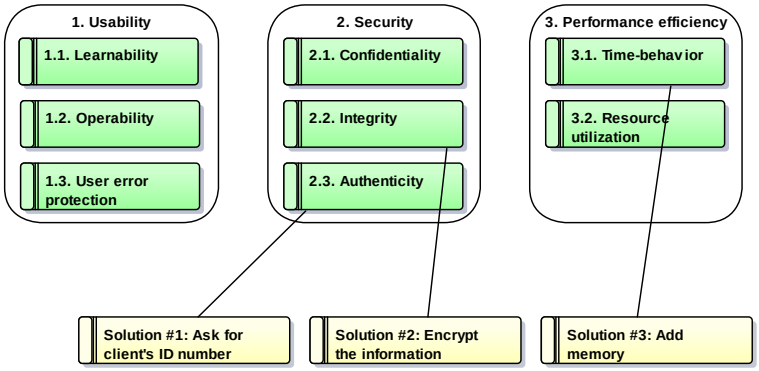

Fig. 4. Selected solutions traced against the quality attributes that they are looking to improve.

In order to analyze the possible impacts of the selected solutions, first, we add the selected solutions into the diagram, and then, we trace them with the quality attributes that they are going to improve. We could chose to *"ask for the client's ID number"* to enhance the authenticity, to *"encrypt the information"* looking to enhance the integrity and to *"add memory"* in order to improve the time-behavior. We use solid lines to trace the solutions to their respective quality attributes (figure 4). Naturally, these solid lines represent positive contributions.

Next, we analyze the impact for each solution on the rest of the quality attributes. If we find some impact, we trace both elements with a dashed line with a stereotype indicating the kind of contribution: positive or negative. In figure 5, we show some of the possible impacts, denoting the positive and the negative ones with lines stereotyped with the corresponding labels.

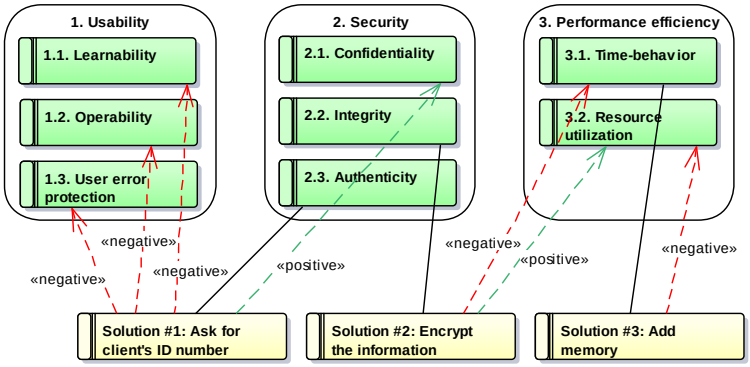

Fig. 5. Positive and negative impacts of the selected solutions.

We have chosen to *"encrypt the information"* in order to improve the integrity, but this solution will negatively impact on *time-behavior* since it will require more time to encrypt and to decrypt the information. However, on the other hand, the solution will positively affect the quality attribute *resource utilization*, as it will need less space to store the information. In the case of solution #3, we can see that it affects both positively and negatively two characteristics of *performance efficiency*, and at the same time it is possible to observe that improving the *time-behavior* with the solution *"add memory"* may result into worse global *performance efficiency* due to the combined effects of both impacts.

4. Minimizing the impact of negative relationships among quality attributes

At the moment, we have just identified the impacts. Now it is time to analyze them objectively, so therefore we should measure the global quality as a result from the application of these selected solutions. Then, we could choose another set of solutions and repeat the process until we have a satisfactory result: to detect the positive and negative impacts due to the contributions among quality attributes and to measure again the global quality of the system.

We will use the Olsina's criteria[5] to measure the impact of our selected solutions, so, we should select a set of metrics according to the different types of solutions. Maybe some of the metrics used to measure the quality attributes could be sufficiently suitable to measure the impacts of the solution. If not, we should configure new ones.

Let us take the solution #, *"ask for client's ID number"*. First, we must create and define a set of attributes for each characteristic traced with the solution. Since solution #1 has connectors against five characteristics, we will define at least one attribute for each of them:

Table 1. Attributes defined for the selected solutions.

Characteristic	Attribute	Definition
Learnability	1.1.1. Learnability time	Time needed to complete the operation.
Operability	1.1.2. Transactions made in one try	Transactions made without needing cancelation or retry
User error protection	1.1.3. Mistakes for ID entry	Transactions made without mistakes on the ID entry
Confidentiality	2.1.1. Transactions made by the account holder	Transaction data without problems of unauthorized access
Integrity	2. 2.1. System modules accessed by authorized people	System ensures cannot have unauthorized access

Next, we must create the metrics to measure each attribute. We will use a reduced template to define the metrics (refer to Olsina's work[5] for the complete template) and here we will define only one direct metric as an example:

Attribute name: 1.1.1. Learnability time

Indirect metric name: Average transaction time (ATT).

Objective: Determine the average transaction time for all the transactions (n).

Function specification: $ATT = \frac{\sum_{i=1}^{n} TransactionTime_i}{n}$

Direct metric name: Number of attempts until average transaction time (TUATT).

Objective: To determine the number of transactions of each specific client (m) with a duration over the average transaction time (ATT).

Function specification: $TUATT = \sum_{i=1}^{m} Transaction_i \forall TransactionTime_i > ATT$

Once we have the metric values for the entire set of attributes impacted for the set of solutions, we can select another set of solutions, we must detect the positive and negative impacts because of the contributions among quality attributes and we have to measure again the same metrics. We will repeat this process until we are satisfied with the value of the whole quality for the system, as proposed by De Florio[6]: "*Measure the effectiveness of the*

attempted solutions, rank them with respect to past solutions, derive and persist conclusions, and update the reactive and proactive models accordingly".

5. Conclusion

There are different models and strategies to measure the quality of software and web applications, and also to identify those characteristics that need to be improved, by means of a consistent set of metrics, some of them urgently while others just in a convenient way. However, maybe all this effort could be vain if the global quality is not improved according to the previous expectations due to the contribution relationships among non-functional requirements. If we configure a consistent set of metrics to measure the global impact of the possible solutions that would be applicable, it would be possible to measure the results of the application of different set of solutions until the desired level of global quality is achieved. This level of quality is equivalent to the "system identity", since the set of non-functional requirements of a system must be coherent with the specifications of that system. Thus, software applications could be adapted to the different situations depending on the values expected for their quality attributes in a given moment.

We are following the studies about how to measure and evaluate software and web application headed by Olsina et al.[1,2,5,7,8,9,10,11] since we have found a very consistent way to measure and to evaluate software and web applications. Miguel et al.[12] have also presented a good review of several quality models that could complement the Olsina's work. Wiegers[3] and Chung et al.[13,14] have worked on the contributions about software quality attributes, particularly the lasts, when they propose a mechanism composed by a technique and a notation to trace the possible solutions against the non-functional requirements. Finally, Mairiza et al.[15] analyze and classify a set of proposals for the identification, analysis and resolution of conflicts among software quality attributes.

References

1. Olsina, L., Papa, F., and Molina, H.: How to Measure and Evaluate Web Applications in a Consistent Way. In Springer HCIS book Web Engineering: Modeling and Implementing Web Applications. Rossi, Pastor, Schwabe, and Olsina (Eds.), pp. 385-420, (2008).
2. Olsina, L., Lew, P., Dieser, A., and Rivera, B.: Using web quality models and a strategy for purpose-oriented evaluations. In Journal of Web Engineering, 10(4), pp. 316-352, (2011).
3. Wiegers, K., J. Beatty: Software requirements. 3rd. ed., Microsoft Press, (2013).
4. ISO/IEC 25010: 2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models (2011).
5. Olsina, L., Lew, P., Dieser, A., and Rivera, B. Updating Quality Models for Evaluating New Generation Web Applications. In Journal of Web Engineering, Vol. 11, No.3, pp. 209-246, (2012).
6. De Florio, V. Antifragility = Elasticity + Resilience + Machine Learning. Models and Algorithms for Open System Fidelity. 1st International Workshop "From Dependable to Resilient, from Resilient to Antifragile Ambients and Systems" (ANTIFRAGILE 2014). Procedia Computer Science 32, pp. 834 – 84, (2014).
7. Becker, P., Papa, F. and Olsina, L. Process Ontology Specification for Enhancing the Process Compliance of a Measurement and Evaluation Strategy. In CLEI Electronic Journal, Vol. 18, No. 1, Paper 2, (2015).
8. Olsina, L., Dieser, A. and Covella, G. Metrics and Indicators as Key Organizational Assets for ICT Security Assessment. In Emerging Trends in ICT Security, pp. 25–44, (2013).
9. Lew, P., and Olsina, L. Relating user experience with MobileApp quality evaluation and design. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8295 LNCS, pp. 253–268, (2013).
10. Olsina, L., Santos, L., and Lew, P. Evaluating Mobileapp Usability: A Holistic Quality Approach. Web Engineering, pp. 111–129, (2014).
11. Lew, P., Olsina, L., Becker, P., and Zhang, L. An integrated strategy to systematically understand and manage quality in use for web applications. Requirements Engineering, 17(4), pp. 299–330, (2012).
12. Miguel, J., Mauricio, D. and Rodríguez, G. A Review of Software Quality Models for the Evaluation of Software Products. In International Journal of Software Engineering & Applications (IJSEA), Vol.5, No.6, (2014).
13. Chung, L., Nixon, B. and Yu, E. Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design. In 1st International Workshop on Architectures for Software Systems, in Cooperation with the 17th International Conference on Software Engineering, ICSE 1995, pp. 31-43, (1995).
14. Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. The NFR Framework in Action. In Non-Functional Requirements in software engineering, Volume 5 of the series International Series in Software Engineering, Springer, pp 15-45, (2000).
15. Mairiza, D., Zowghi, D., and Nurmuliani, N. Managing conflicts among non-functional requirements. In 12th Australian Workshop on Requirements Engineering (AWRE '09), (2009).