Comparing internal and external software quality measurements

Dimitris STAVRINOUDIS\(^a\) and Michalis XENOS\(^a,1\)

\(^a\) School of Sciences and Technology, Hellenic Open University, Patras, Greece.

Abstract. Modern software development companies that have a quality assurance program use measurements and standards to improve product quality as perceived by the users of these products. However, during the entire software life cycle, except for the final customers, different types of ‘users’ also appear. This paper firstly shows the different views of software quality of these types of users. It also presents the internal and external measurement methods that we used in order to measure the users’ opinion of software quality, the benefits and drawbacks of each method, as well as information concerning the techniques used to conduct internal and external measurements. Surveys and examples showing whether software metrics and external views of quality are correlated are also presented. The aim of this paper is to determine up to what point and in which cases can we rely on software metrics in order to define the users’ perception of software quality.

Keywords. Software quality, Software metrics, User Satisfaction Measurements

Introduction

In almost all development procedures the end-product quality is strongly connected to measurements that are used to monitor quality factors and to serve as an early indicator of erroneous development. International standards, such as ISO 9001 [1] and the guideline for software development ISO 9000-3, emphasize the need of measurements for assuring product quality. The Capability Maturity Model (CMM) [2] focuses on the need to conduct measurements during software development. As expected, due to their nature, although such standards and models pinpoint the need of software measurements, they do not provide specific guidelines on how to conduct such measurements and what should be aimed in a measurement program. In this paper we present experiences from 10 years of research in the field of software metrics and measurements, discussing how a measurement program is set up, what the goals of such a program should be and how various types of metrics could be used.

Software measurements can be used to prevent catastrophes by offering early warnings of erroneous development, to prevent errors and defects, not only in the early stages of a product’s release but even before, and to aid in monitoring software development. A common distinction is to separate measurements into internal and external, as we discuss further in the following sections. Internal measurements are the ones conducted automatically on program code by the use of internal software metrics, while external are the ones that require the end product and –in many cases– the users’

\(^1\) Corresponding author. 23 Sachtouri Str, GR 26222, Patras, Greece. Tel.: +30 2610 367405; fax: +30 2610 367416. E-mail address: xenos@eap.gr
involvement. While internal measurements can be used to aid in early prevention of errors and defects [3], external measurements can be used to calibrate internal measurement tools and to provide users’ perceived measurements of software quality. Although such measurements are subjective, they are accurate indications of users’ opinion of software quality. After all, one should never neglect that quality indicates the satisfaction of the end-user needs.

1. Views of software quality

It has been broadly accepted that quality is an abstract concept and cannot be easily understood as a single entity in order to specify measurable elements clearly and directly. Thus, it has been broken down into a number of sub-components or characteristics. These quality characteristics constitute the quality of a product and the overlap among them is the minimum possible. In order to comprehend the meaning of quality and measure it with a systematic method a number of models have been proposed, where the quality characteristics and how they relate to each other are described. An analogous standard that has been proposed and is commonly used is the ISO9126 standard [4]. According to ISO9126, software quality could be broken down into six major characteristics, which are functionality, reliability, usability, efficiency, portability and maintainability and these quality characteristics are split to a number of quality sub-characteristics. The above mentioned product quality characteristics can be divided into two different sets: external and internal. Customers care about external quality characteristics such as: functionality, reliability, usability, efficiency, flexibility, friendliness, simplicity, etc. simply because these are the characteristics which can easily be seen by the use of the product. On the other hand, developers care about internal quality characteristics such as: maintainability, portability, reusability, testability, etc. because these characteristics relate to their development efforts.

However, in a software quality assurance program the term ‘user’ is not clearly defined. It is assumed that the user is the customer who receives the software from the production company. But there are more types of ‘users’ during the entire software life cycle. For example, the testing department of a company uses the software before its final release and expects it to meet a set of quality criteria. Similarly, the maintenance department may also be considered as users. Furthermore, companies reuse parts of the code and software components that they had either built previously or procured (obtained) from other companies. In other words, almost everyone that participates in the production plays the role of the user at least at some level.

Despite new methods and techniques being applied to the production process, all such users are still having problems with software of unacceptable quality. Quality is in the eye of the beholder and is judged according to sets of desired characteristics, which hold some significance for the various stakeholders. As a result, different views of software quality may be observed, according to the weight given to each quality characteristic. In other words, there are differences in the perception of factors related to quality as perceived by customers and as perceived by the users within the development team. As it was mentioned earlier, the former are interested more in internal characteristics, whereas the latter are interested more in external ones.

However, similar differences may be observed between users of the same type (e.g. customers), according to how they focus on quality characteristics. For example, one user may emphasise more on efficiency and another more on functionality. The
possible weight given to each of the quality characteristics depends mainly on the type of the software application, how it was produced, for whom and possibly even why. Such questions relate to the users and their requirements. The appearance of different perceptions even of an internal quality characteristic is observed in a development team, according to the specific role of each member of this team. Since the phase during software life cycle that demands the most effort and cost is maintenance (sometimes this cost may exceed the 60% of the total cost of software development) [5], it is interesting to examine the various opinions of maintainability.

2. Measurement methods

As previously mentioned, software quality characteristics can be divided into two different sets: internal and external. Similarly, two different types of measurement methods have been developed according to the aforementioned sets of quality characteristics. In our research we used both internal and external measurements, since these two types are supplementary to each other. In other words, neither of them can completely substitute the other, because of the differences in their goals. In the internal measurements the researcher focuses on the attributes that can be measured purely in terms of the software product, by examining the product on its own, separate from its behaviour. Here, software metrics are used that measure internal product characteristics, without conferring with the users. On the contrary, external measurements are direct measures of external product quality characteristics by questioning the end-users. The researcher focuses on the attributes that can be measured only with respect to how the product relates to its environment. Here, the behaviour of the process is important, rather than the entity itself [6].

What one can obviously derive from the above definitions is that the internal measurements can be applied in a very fast, easy and automated way. Moreover, the implementation of appropriate software tools that can give the results of software metrics is efficient. As a result, the error frequency during this process is minimal and unconsidered and the results of these measurements are considered to be objective and valid. The cost of internal measurements is significantly low, since what is really measured consists of tangible characteristics of the source code of a program. Their results are easily analysed with statistical methods. Furthermore, internal measurements may be applied even before the completion and final release of a product. Consequently, in this case, corrective actions may be followed during the implementation of a software product, before its final release to the customer. However, internal measurements usually do not offer high-level information about the quality of a software product. In other words, the data that derive from them may be difficult to be interpreted and utilised, because of their weakness to be directly related to the external quality characteristics of a software product.

On the contrary, the results derived from external measurements can be interpreted and used directly by the software production company, because they measure straightly the quality of a software product according to these desired external characteristics. External measurements are significant in every software quality program, since all the quality standards emphasize the measurement and the evaluation of the opinion of the end-users. Moreover, the quality data derived from these measurements can be directly analysed, which is a proper fact for software production companies with a high maturity level. Besides, external measurements keep step with the main aim of
software quality and contribute to the improvement of the public image of a company. Finally, the role of the customer during the production phase is promoted, since not only his/her opinion is taken into great consideration, but he/she also participates effectively to the software production process. However, the results from external measurements may be liable to disputes, because they are based on the subjective opinion of users. Here, the error frequency is significantly greater than in the case of internal measurements. In other words, it is difficult to analyse these measurements, due to high error rates and various data scale types. The cost of conducting this kind of survey is very high, because usually it is inefficient to automate it and because human resources in this kind of survey are necessary.

2.1. Internal measurements

Internal measurements can be applied in an automated and cost effective way. Software companies that conduct these measurements use appropriate methods based on software metrics. Numerous software metrics related to software quality assurance have been proposed in the past and are still being proposed. Furthermore, several books presenting such metrics exist such as Fenton and Pfleeger’s [6], Shepperd and Ince’s [7] and others. As software metrics may be measured automatically, a number of measurement tools have already been implemented. The use of such tools reduces the cost of the measurements and the do not need further inspection for their correctness. In our research we used the software measurement and metrics environment ATHENA [8]. It is a metric-free and language-free metrics environment, which provides completely automated measurements and therefore, the collection of raw data was effortless, as it is required in any enterprise that collects developer-oriented measurements. The selection of the appropriate metrics for internal measurements of one specific product depends on the nature of this product and the programming language used during its implementation. Traditional, broadly used metrics can always be applied regardless of the programming language. The use of internal software metrics is meaningful only when they are applied to large fragments of code. Measurements must not be restricted only to small parts of a software program in order to draw useful conclusions. However, using a framework of metrics allows the programmers to locate modules and routines with a low level of quality. The use of metrics assists the programmers to inspect their code and make the necessary improvements during the implementation phase.

Although in our research we used a large number of metrics, for the presentation of this paper we have selected metrics that can be applied in every routine and every module. In this way, i.e. if we are not restricted by factors such as the specific type of programming languages or programming style or even the type of the application that is being implemented, this paper will be able to lead to generalised and language independent results and conclusions. As a result, we have chosen Halstead’s software science metrics [9], McCabe’s cyclomatic complexity [10] and Tsai’s data structure complexity metrics [11], which are a set of kernel metrics and are representative ones in order to measure size, data complexity and data structure complexity respectively. However, when programming in a specific language or framework of a specific programming style, a software production company must choose an appropriate set of metrics that can also be applied to this style. For example, in object oriented programming, besides the commonly used metrics mentioned above, the developers should also use a set of desired object oriented metrics [12]. Similarly, in web engineering different measurements must also be conducted, as the literature identifies...
unique characteristics of web systems that reflect technical, usability and organisational issues [13]. Since in our research different kinds of applications were measured, these specific metrics could not be applied to all of them. As a result, we focused mainly on the commonly used metrics, in order to draw meaningful conclusions.

2.2. External measurements

Most common external evaluation techniques are the heuristic evaluation by experts [14], the performance measurements [15], [16] and the user-perceived software quality measurements [17]. The heuristic evaluation requires a number of 4-5 experts on the specific software type and offers subjective inspection (not measurements). Performance measurements require both experts and users willing to participate at on-site evaluation (performed in a usability lab), thus are expensive and not appropriate in cases of distributed users. Due to all these facts, the external measurements presented in this paper have been conducted using user-perceived quality measurements. This kind of measurements were based on the $QWCO_S$ technique [17] (Qualifications Weighted Customer Opinion with Safeguards technique) that is shown in Eq. (1).

$QWCO_S$ technique estimates the result of the external measurements (EM) and is based on the $O_i$ (normalised score of customer $i$ opinion), the $E_i$ (qualifications of customer $i$) and the use a number of control questions –also called safeguards– where $S_T$ is the total number of control questions, from which the customer $i$ (from a total of $n$ customers) has replied correctly at $S_i$.

\[
EM = QWCO_S = \frac{\sum_{i=1}^{n} \left( O_i \cdot E_i \cdot \frac{S_i}{S_T} \right)}{\sum_{i=1}^{n} \left( E_i \cdot \frac{S_i}{S_T} \right)}
\]

Although such external measurements cannot be automated as internal measurements are, effort can be made on the automation of the data collection, the testing of the validity of the responses (using the aforementioned technique) and the analysis of the results (that is a typical statistic analysis). For this purpose the $CESM^2$ Tool [18] has been used. $CESM$ Tool allows the use of a measurements technique. The $CESM$ is comprised of the Questionnaire Designer, the surveys web pages, the Questionnaire Analyser and a database. The author of the questionnaire has to specify which user–oriented quality characteristics derived from the (included in the tool) ISO9126 classification will be dealt with. The questions of the questionnaire must be clustered in groups, according to which quality characteristic they refer to. Each group of questions can be given a different weight, depending on the emphasis given to the corresponding quality characteristic by the author.

3. Results from the surveys

Modern software developers that have a quality assurance program use metrics and standards to improve product quality as perceived by the end-users of the product. After all, the users’ view of quality is usually considered to be paramount. According

\[^2\] $CESM$ is an acronym for: ‘Collecting External Software Measurements’.
to Fenton’s axiom [6], good internal structure is expected to provide good external quality. However, the main question that arises is up to what point and in which cases may a software production company rely on the use of metrics in order to guarantee a priori a satisfying score in the external measurements of its products and, in other words, acceptance by the users. Furthermore, another research question is when and under which circumstances a low score in these metrics indicates a possible failure of the software product’s quality, as this will be perceived by the users. Our basic aim was to identify cases that users accept as having high quality, with early-measurable internal metrics as potential indicators. In the following sections two different surveys are presented. The survey in section 3.1 examines the correlation between internal measurements, which were conducted as described in section 2.1, and external measurements that indicate the end-users’ opinion of quality. The survey in section 3.2 examines the correlation between similar internal measurements of software products and external measurements of the opinion of programmers which would have to maintain or reuse components of these products.

3.1. End-user survey

This presented survey was based on end-users and included data from $N=34$ components of software products developed for various projects. Each of these $N_i$ ($i \in \{1, 34\}$) components could be measured as a stand-alone product, therefore it could be evaluated separately to other components. All these components were assessed using external measurements. The results presented in this section are based on external measurements from $K_i$ users for each component $N_i$. The number $K_i$ varied for each component $N_i$ and applies that $K_i \in \{35, 56\}$. End-user survey data were combined in a single measurement result for each component, using the Eq. (1), thus resulting a single number that combines the external measurements for component $N_i$. Due to the nature of the result of the Eq. (1), which hereafter will be represented as $EM_i$ for the component $i$, it applies that $EM_i \in [0, 1]$.

Since we found a very high correlation between the results of the internal metrics that we used, we focused on the language level $\lambda_i = N \log_2(n)(2n_2/n_1N_2)^2$, which measures how well the programming language is used, on the essential size ratio $R = (n_1 \log_2 n_1 + n_2 \log_2 n_2)/N$, which detects if the code contains any code impurities (where $n_1$, $n_2$ is the number of distinct operators and operands, $N_1$, $N_2$ the number of total operators and operands, $N = N_1 + N_2$ is the program size and $n = n_1 + n_2$ is the vocabulary), on the cyclomatic complexity $V(g) = e-n+2$ proposed by McCabe (where $e$ is the number of edges and $n$ is the number of nodes) and on data structure complexity proposed by Tsai. Laying a greater weight on larger routines, the language level of the whole component can be defined as $\lambda = (\Sigma N \lambda_i) / \Sigma N_i$ (where $N_i$ is the program size and $\lambda_i$ is the language level of the routine $i$). Equivalently for cyclomatic complexity the factor $10/V(g)$ was used, where $10$ is the proposed by McCabe maximum complexity, and the measure of the cyclomatic complexity of the whole component can be defined as $V(g) = (\Sigma N_i V(g_i)) / (\Sigma N_i)$ (where $V(g_i)$ is the cyclomatic complexity of the routine $i$). Finally, from data structure complexity the factor $1/(T+1)$ was used, where $T$ is the greatest degree of the polynomials of the component’s routines that derive from the Tsai’s formula $C(K) = \Sigma(C(K_i)) + S(K)$ and $S(K) = (v+e)x^L$ (where $v$ is the number of nodes, $e$ is the number of edges and $L$ is the number of simple circular paths into the reduced data graph). In the derived polynomials, as the degree of a polynomial grows,
the data structure complexity of the program also grows. So the factor \(1/(T+1)\) is disproportional to data structure complexity. In order to summarize all the above mentioned metrics, a combined formula \(\text{IM}_i\) for each component \(i\) was formed, which is shown in Eq. (2). It must be made clear that the above formula does not measure a physical quantity of the component, but is both a customized collective formula created especially for this particular case study and a means by which the results of all the metrics can be combined. The purpose of \(\text{IM}\) is to facilitate the presentation of the results providing an estimation for the overall performance of each component regarding to the metrics used.

\[
\text{IM} = 0.2 \lambda + 0.2 R + 0.4 \left( \frac{10}{V(g)} \right) + 0.2 \left( \frac{1}{T+1} \right)
\]  

(2)

Both external measurement data (Normality test Shapiro-Wilk’s Significance level = 0.857) and internal measurement data (Normality test Shapiro-Wilk’s Significance level = 0.901) were distributed normally. Correlations among the internal metric results and the external measurements were found to be equal to 0.819 which was statistically significant. The scatter plot shown in Figure 1 illustrates this measured correlation. The external measurements are shown on the horizontal bar, whereas the internal measurements are shown on the vertical bar. The line on the scatter plot shows where all points should be in case these two variables were 100% correlated. It is obvious that components with low score in the internal metrics will also have low score in external measurements. Besides there is no measured component, which has low score in the internal metrics while having high, or even average, score in the external measurements. On the contrary, high internal metric results do not necessarily imply high external measurements results. As the scatter plot shows, there are many components that score high in internal metrics, but score poorly in external measurements.

![Figure 1. Correlation between IM and EM](image)

Therefore, the derivative of this survey is that poor internal metric results in most cases would result equally poor results in the external measurements. Consequently, it
is generally acceptable to set a limit to carefully selected internal metrics in order to preserve the external quality of a component as perceived by its users. But, there is no reliable method to find out if the components with high internal metrics score will also have an equally high score in external measurements. Internal metrics can serve as a good first indication for user perceived quality, but they cannot always guarantee successful external measurements as well.

### 3.2. Inside the company survey

Software maintainability is a difficult factor to quantify. However, it can be measured indirectly by considering measures of design structure and software metrics. It is also claimed that logical complexity and program structure have a strong correlation to the maintainability of the resultant software [19]. This survey presents the relation between software metrics and maintainability and was based on the opinion of future maintainers. The survey included data from \( N=29 \) components of software products developed for two large software projects. Similarly with the previous survey, each of these \( N_i \) (\( i \in [1, 29] \)) components could be measured as a stand-alone product, therefore it could be evaluated separately to other components. During the software life cycle these components had to be maintained or even modified in order to be reused in future projects. The results presented in this section are based on external measurements from \( P_i \) programmers for each component \( N_i \). The number \( P_i \) for each component \( N_i \) varied from 28 to 54. The opinions of these programmers were combined in a single measurement result for each component, using the Eq. (1), thus resulting a single number \( EM_i \) that combines the external measurements for component \( N_i \). It is obvious that, as mentioned in the previous survey, that \( EM_i \in [0, 1] \). In this survey 4 outliers were observed and weren’t taken into consideration, since they consisted of either declaration routines or very short routines without any control flow statements that were simply calling other routines.

![Correlation between IMi and EMi](image)

**Figure 2.** Correlation between IMi and EMi

In a similar manner to the external measurements, the results from the internal metrics were analysed and compared to the external measurements. Moreover, the
same combined measure \( IM_t \) was used. Both external measurement data (Normality test Shapiro-Wilk’s Significance level = 0.439) and internal measurement data (Normality test Shapiro-Wilk’s Significance level = 0.732) were distributed normally. Correlations among the internal metric results and the external measurements were found to be equal to 0.730 which was statistically significant. The scatter plot shown in Figure 2 illustrates this measured correlation.

From the results of this survey it was obvious that software metrics’ score and programmers’ opinion of maintainability were highly correlated. In detail, components with high internal measurements scored equally high in external measurements. Additionally, components that failed in software metrics scored poor according to the programmers’ opinion of maintainability.

Furthermore, it was also observed that some components that failed in metrics scored with a great variation according to the programmers’ opinion of maintainability. In other words, low internal measurements do not necessarily imply low external measurements. This variation was observed mainly because of the nature of the application these modules belong to. This observation is in conflict with the relation between software metrics and the customers’ opinion of quality, as it was shown in the survey of section 3.1. In the case of customers, metrics can only detect possible causes for low product quality, whereas satisfaction of internal quality standards does not guarantee success in fulfilling the customers’ demand on quality. On the contrary, in the case of the maintenance phase, good internal structure implies high score of maintainability as this will be perceived by the programmers, whereas low internal measurements do not necessarily entail low score of maintainability. Therefore, the derivative of this survey is that poor internal metric results in most cases would result equally poor results in the external measurements. Consequently, it is generally acceptable to set a limit to carefully selected internal metrics in order to preserve the external quality of a component as perceived by its users.

4. Conclusions

Software metrics are able to offer a very good first indication for external quality. It was shown in both presented surveys that they are highly correlated with external quality characteristics. Specifically, they provide an easy and inexpensive way to detect and correct possible causes for low product quality as this will be perceived by the customers. Setting up measurement programs and metric standards will help in preventing failures to satisfy customers’ demand for quality. Programmers should keep the metrics score of their programs within acceptable limits. This can be accomplished by the use of appropriate programming environments that offer basic quality assurance functions, such as setting a source code measuring program, and define basic metrics and their limits. These environments may also offer comparisons between different versions of the same project, allowing the programmers to monitor the progress of their performance in relation with the measurement goals they have set or with the requirements of the quality assurance program being followed.

Generally, the better results the metrics achieve when measuring software modules, the higher level of maintainability according to programmers opinion these modules will have, so the less effort will be spent during the maintenance process. This conclusion was also observed when measuring different versions of the same component, where the subsequent ones achieved better results in internal measurements.
than their former. Consequently, sometimes it is preferable for a software development company to risk not only to correct, but also to rewrite specific parts of working code in order to decrease the effort during the maintenance phase and to avoid or early cure possible customers’ problems. However, if a component fails in internal measurements it is preferable not to reject it automatically although it usually fails in external measurements, too. On the contrary, as low score in metrics sometimes is expected, because of the nature of what a component may implement, the programmers should inspect carefully their code and decide whether it needs to be corrected (or even re-written) or not.

Acknowledgement

The work presented in this paper for the last 4 years is funded by the project: “Support of Computer Science Studies at the Hellenic Open University” (2003-2008, Greek Ministry of Education: Operational Programme “Education and Initial Vocational Training”).

References