Towards Introducing Execution Tracing to Software Product Quality Frameworks

Tamás Galli, Francisco Chiclana, Jenny Carter, Helge Janicke

Centre for Computational Intelligence, Faculty of Technology, De Montfort University, The Gateway, Leicester, LE1 9BH, United Kingdom

p10553741@myemail.dmu.ac.uk, chiclana@dmu.ac.uk, jennyc@dmu.ac.uk, heljanic@dmu.ac.uk

Abstract: Execution tracing and logging significantly influence the time spent on localizing software errors; consequently, they have essential impact on maintainability. Moreover, in certain situations these tools are the best suited instruments to analyse the behaviour of distributed, multithreaded or embedded applications. In spite of this, software product quality frameworks do not include execution tracing or logging as a quality property. In this paper we examine the extension possibilities of the present software product quality frameworks to accommodate execution tracing. In addition, the scope of the investigation includes facilities of the frameworks to address the uncertainty involved in quality measurements.

Keywords: quality frameworks; execution tracing; logging; uncertainty

1 Introduction

Execution tracing and logging are frequently used as synonyms in software technology; however, the first one rather serves the software developers to localize errors in applications, while the second one contributes to administration tasks to check the state of software systems [12], [20], [28], [30], [35]. In the scope of this paper we also use the two phrases as synonyms.

Execution tracing dumps the data about the program state and the path of execution for developers for offline analysis, which helps to investigate error scenarios and follow changes in the state of the application. Thus, execution tracing and logging belong to dynamic analysis techniques i.e. testing, investigating live systems, which are integral parts of the maintenance activities. Dynamic analysis techniques can be applied only if the software is built and executable in contrast to static analysis techniques. However, both methods are applied to achieve the same goal of diagnosing errors, with each technique having its own particular advantages [5], [8], [11], [41].

Spillner, Linz, Schaeffer in [37] make distinction between two types of software maintenance: (1) corrective maintenance, the purpose of which is eliminating errors in the software and (2) adaptive maintenance to change the software according to new requirements. Both kinds of maintenance necessitate analysis methods to find errors but this activity dominates in corrective maintenance. The proportion of maintenance costs in the whole software life-cycle amounts to a large part [1], [25], [26], thus decreasing the time devoted to localizing errors can therefore decrease the maintenance costs.

The increasing size and complexity of software systems makes localizing software errors more difficult. This difficulty is aggravated by the enormous number of software and hardware combinations. Adding execution tracing to key places of the application can drastically reduce the time spent with debugging [3].

Utilizing a debugger is time consuming and does not offer adequate solution

- if performance problems have to be resolved because debugging the source code considerably changes the environment from point of view of execution performance. Moreover, performance is sensitive to external workload, configuration parameters, underlying hardware and software components [3].
- in case of real-time, embedded systems as it might be harmful or impossible to reproduce the error e.g. in control applications [39].
- in the case of concurrency, as it changes the race conditions for parallel running execution threads or processes. In addition, multi-core systems also need to be considered which may even have multiple clock domains [39].

A wide survey on concurrency [10], for which 10% of all Microsoft employees from development, test, and program management were selected, also supports that analysing concurrency faults makes up a significant part of their correction costs. 66% of the respondents had to deal with concurrency issues. The reproduction of these issues was classified in a five categorical scale ranging from easy to very hard. 72.9% of all responses classified reproduction of concurrency issues in the two most difficult categories. Moreover, the respondents stated that the severity of these issues, qualifying on an ordinal scale with four categories ranging from least severe to most severe, belongs to the top two: most severe, and severe. In addition, 65% of the respondents expressed the future expectation that the concurrency issues would be more problematic.

Laddad states in [27] that execution tracing is the only adequate tool to help with the analysis of run-time errors in the case of distributed systems and multithreaded applications. In the case of embedded applications, which have no user interface, by means of tracing the developer or system maintainer can answer questions such as what the application is doing [39].

Diagnosing regression test errors and finding root causes implicate major difficulties. Fault localizations can be grouped in three categories [32]: (1) dynamic dependence analysis of the failing program execution, (2) comparison of the failing program execution with a set of error free executions, (3) comparison of the failing program execution with a program execution which does not manifest the error in analysis. Variants (2) and (3) are based on execution tracing.

An experiment conducted by Karahasanovic and Thomas [21] categorized the difficulties related to the maintainability of object-oriented applications. Program logic was ranked the first in the source of difficulties. Understanding the program logic belongs to the category of software specific knowledge which can greatly be enhanced by execution tracing, offering a basis for trace visualization and program comprehension [36].

Tracing, logging or constraint checks represent significant parts of the source code of applications. Spinczyk, Urban and Lehmann [38] state that the ratio of code lines related to monitoring activities such as tracing, logging reached approximately 25% in their measurements of commercial applications. This ratio shows that a significant amount of source code is written to deal with execution tracing, which in itself is an important quality factor. However, execution tracing does not need to be tightly coupled to the application code and can be localized in separate modules [27], [31], [40].

All the above indicate that execution tracing and logging have essential impacts on the analysability of software systems. In certain cases these tools are inevitable to localize errors or investigate software behaviour. Nevertheless, present software product quality frameworks do not exhibit any property to describe execution tracing but they usually offer the potential to be extended. In this paper we analyze such extension points and articulate concrete possibilities for extension in the context of the current investigation. Software product quality frameworks form complete models to support the description and assessment of the quality of software products. As research shows [23], conformance with process quality models does not guarantee good-quality software products, motivating the application of software product quality frameworks in synergy with process quality models.

In addition, measuring software product quality is difficult. Some quality measure elements are easier to measure than others even if the quality measure element is well defined [33]. All of the software product quality frameworks reviewed (Section 2) include the description of qualitative properties in a quantitative manner and quality measure elements which cannot be measured directly but only derived from the observation of the behaviour of software developers, maintainers, operators and users. This condition introduces uncertainty in software product quality frameworks, which has recently been admitted and accepted by defining the subjective measurement method category in ISO/IEC 25021:2007:

"Subjective measurement method - Subjective measurements are those where quantification is influenced by human judgement. Subjective measures are used when no formal objective procedures of measurement can be applied. The value of the quality measure element is influenced by human judgement as an evaluator. Therefore it is necessary to interpret the results with respect to the number of evaluators and statistical methods used for the measurement result calculation. Both should be stated while presenting the measurement results."

Manifestations of uncertainty can be classified into three broad categories: (1) objective uncertainty that refers to the future, (2) subjective uncertainty that refers to the future, and (3) subjective uncertainty that does not refer to the future but helps to categorize elements [6], [24]. Category 1 is modelled by the classical probability theory, while category 2 is considered as an application area of Bayesian statistics. Category (3) on the other hand, is modelled and studied under the name of fuzzy logic. Thus, we also aim to examine in the scope of the research how far the current quality frameworks can ensure the link to quality measures described by means of fuzzy logic to consider the above subjective uncertainty. The authors have already presented a pilot study on modelling execution tracing quality by type-1 fuzzy logic [9].

Summarising the above, the main contributions of the paper, which will be elaborated in detail in the following sections, refer to: (1) the need for execution tracing quality to be appropriately implemented within software product quality frameworks; (2) the significant differences between the current software product quality frameworks to allow such implementation; (3) the ability of the ISO/IEC software product quality frameworks to provide mathematical computations to define metrics and measures, which can be exploited in capturing and implementing subjective uncertainty within their quality models; and finally, (4) the outline of metrics and a measure for execution tracing quality for both the ISO/IEC 9126-1 and ISO/IEC 25010 software product quality frameworks.

The rest of this paper is structured as follows: Section 2 introduces the software product quality frameworks. Section 3 demonstrates the extension facilities of these frameworks, while section 4 presents a discussion on the particular changes in the frameworks required to encompass execution tracing and finally the last section closes with the conclusions.

2 Software Product Quality Frameworks

The analysis of quality frameworks was conducted, using IEEE, ACM and EBSCO databases, to discover existing alternatives or predecessors to describe software product quality. The investigation focused on software product quality models, which describe the whole set of software product quality; therefore we refer to them by the term software product quality frameworks.

In the current frameworks [13], [14], [23], [29], if the traceability property is present it refers to requirement traceability, i.e. how requirements can be followed during the development of the software. Some quality metrics and measures defined in ISO/IEC 9126-2:2003, ISO/IEC 9126-3:2003 and in ISO/IEC 25021:2007 show overlapping and similarities to execution tracing but their definitions are ambiguous and difficult to approach from a practical point of view: e.g. calculating the ratio of the number of diagnostic functions and the number of necessary diagnostic functions. Such metrics are associated with the Internal and External Analysability sub-characteristic of the characteristic Maintainability: (1) Activity Recording, (2) Readiness of Diagnostic Functions, (3) Audit Trail Capability, (4) Diagnostic Function Support, (5) Failure Analysis Capability, (6) Failure Analysis Efficiency, (7) Status Monitoring Capability [17], [18]. This means that the existing frameworks do not consider execution tracing as an aspect of software quality.

2.1. Early Frameworks

Early software product quality frameworks appeared in the second half of the 1970’s to assess quality and show the way for improvements in software products [2], [29]. These frameworks had a significant influence on the recent software product quality frameworks published by the ISO standards. They kept the hierarchic nature abstraction of quality.

2.1.1. Software Product Quality Model of Boehm, Brown and Lipow

The first complete model to assess software product quality was developed by Boehm, Brown, and Lipow [2]. They established a set of quality properties, which they call characteristics, and one or more metrics to each of them. They defined the notion of metric as (1) a quantitative measure that describes the degree to which the software product possesses the given characteristic, and (2) the overall software quality must be able to be described by the function of the values of the metrics.

They came to the conclusion in their study that establishing a single overall metric for software product quality would implicate more difficulties than benefits because many of the major individual quality characteristics are conflicting; moreover, the metrics they associate to the quality characteristics are incomplete measures of the quality characteristics. Therefore, they developed a hierarchical model. The hierarchy comprises of eleven high-level characteristics representing different aspects of software product quality [2]: (1) understandability, (2) completeness, (3) conciseness, (4) portability, (5) consistency, (6) maintainability, (7) testability, (8) usability, (9) reliability, (10) structuredness, and (11) efficiency.

The model is language-independent and independent of programming paradigms, however many metrics were tested on the structured language Fortran. Additional metrics to the published ones can easily be defined and the model offers possibilities for tailoring.

2.1.2. Software Product Quality Model of McCall, Richards and Walters

McCall, Richards, and Walters published a different framework [29] to Boehm's model [2] to assess software product quality. The authors describe a global view of software product quality as a combination of three distinct activities: (1) product operation, (2) product revision, and (3) product transition i.e. the description considers also process related properties. The objective of their investigation was to provide a concept to acquisition managers to specify and measure quality in a quantitative manner in software products related to air force applications.

They established a set of software quality properties that describe the overall quality of the software product and they named these properties factors. The quality factors they associated with criteria. Criteria are attributes of the software or software development process by which the factors can be judged and defined. A criterion can have sub-criteria in a hierarchical manner and one criterion may affect more quality factors. The criteria are coupled with metrics that make possible the measurement of the criteria or sub-criteria. The separation between properties that would also qualify for being both criterion and factor the authors made the decision: user-oriented properties are quality factors while software-oriented are criteria.

Quality farctors:

(1) Product operation: (a) correctness, (b) reliability, (c) efficiency, (d) integrity, (e) usability; (2) Product revision: (a) maintainability, (b) testability, (c) flexibility; (3) Transition: (a) portability, (b) reusability, (c) interoperability;

The authors also investigated the impact of the quality factors on each other, i.e. if a particular factor is present with a high degree of quality what quality is expected for the other factors. Beside the positive relationships, there exist also negative ones between some quality factors. In those cases finding a compromise is crucial, e.g. integrity and interoperability conflict with each other, which means that the more interoperable the system is the more difficult it is to keep its integrity.

Similarly to the previous framework, this model is language-independent because its metrics are language independent and independent of programming paradigms. It leaves room for extension and tailoring.

2.2. Recent Frameworks

Recent software product quality frameworks appeared after 1990. They can be divided into three categories on the basis of their philosophy [7], [13], [14], [22]: (1) hierarchic models of the ISO/IEC standards which are strongly influenced by the early frameworks, (2) adaptations of the ISO/IEC standards, and (3) the non-hierarchic framework of Dromey. Their presentation follows in historical order.

2.2.1. Software Product Quality Model of ISO/IEC 9126 Standard Family

The quality model of the ISO/IEC 9126 standard family comprises of a hierarchic model for software product quality and quality in use. The first version of the standard was issued in 1991 which was superseded by the next version issued in 2001 [14]. The description of software evaluation was moved from the second version to the multipart standard ISO/IEC 14598 [15]. The standard ISO/IEC 25010:2011 [13] revised the quality models described by the ISO/IEC 9126 standard family.

Terminology:

- Quality characteristics: high-level quality properties which are located at the top of the hierarchy. In the terminology of ISO/IEC 14598 standard family they are called attributes.
- Sub-characteristics: Quality characteristics which are located somewhere in the hierarchy but not at the top-level. Sub-characteristics are always assigned to a higher level characteristic or sub-characteristic.
- Quality metrics: Definition of the measurement method of quality properties including the definition of the measurement scale. Quality metrics are assigned to sub-characteristics or characteristics.
- Internal quality metrics: Metrics whose inputs are formed by the intrinsic properties of the software product.
- External quality metrics: Metrics which cannot be measured directly but only derived how the software relates to its environment.
- Quality of use: The user's view of quality.

Concepts:

The standard ISO/IEC 9126-1:2001 defines three basic views of the quality: (1) internal view, (2) external view, (3) user's view. Internal view of the quality means the quality measured by the internal quality metrics. This reflects the quality of the source code or documentation. It is very useful if the software product is not developed as far as it could be tested. The external view of the quality is measured by the external metrics. It shows how the product relates to its environment. The user's view of the quality is illustrated by the quality in use reflected by the quality in use metrics.

Internal and external metrics either need to be in cause-effect relationships or they need to correlate with each other. This is called predictive validity i.e. from the measurement by the internal metrics conclusions can be drawn relation to the external metrics and external quality of the software.

The software product quality model introduces six high-level characteristics: (1) functionality, (2) reliability, (3) usability, (4) efficiency, (5) maintainability, (6) portability. In addition to their sub-characteristics, each of these characteristics has an internal and external variant to form an internal and external model.

The quality in use model has four high-level characteristics without sub-characteristics: (1) effectiveness, (2) productivity, (3) safety, (4) satisfaction. External metrics need to have predictive validity for the quality in use metrics.

The model is language-independent and independent of programming paradigms.

2.2.2. Software Product Quality Model of Dromey

Software does not directly display quality properties but it shows product properties, which contribute to the quality properties in a positive or negative way. Dromey argued that the previously published software product quality models adequately addressed these particularities. He proposed a model where the main focused was on the product properties, which he calls quality-carrying properties, and on the relationship between product and quality properties in a non-hierarchic manner [7].

Terminology:

- Quality attribute: high-level quality property
- Structural form: programming language constructs
- Quality-carrying properties: binary-value variables which determine the quality

As quality attributes Dromey identified the six high-level quality characteristics of the ISO/IEC 9126:1991 standard and extended this set with the attribute reusability [7].

Concepts:

The model makes possible the specification and analysis of the relationships between quality attributes, quality-carrying properties, and structural forms. The bottom-up approach facilitates for developers to specify or investigate which quality-carrying properties be associated to the structural forms of a particular application. The top-down approach facilitates for designers to specify the quality requirements and attributes the software needs to satisfy and identify the quality-carrying properties for the structural forms to fulfil the quality needs [7].

Quality is depicted by the relations: (1) between quality-carrying properties and quality attributes; (2) between quality-carrying properties and structural forms. Dromey also proposes profiles for both relations. Quality-carrying properties and structural forms have precedence rules in such profiles. If the precedence rules are kept, the model is able to classify software quality defects.

The basic mechanism of the model can be formalized as (1) if each quality-carrying property of a structural form is satisfied, then that structural form will have no quality defect; (2) if a quality-carrying property of a structural form is violated, then it will contribute a quality defect to the software.

The definition of the model is language dependent in contrast to the previously presented software product quality frameworks because it uses programming language level constructs as structural forms and their properties as quality-carrying properties. It was prepared for supporting the procedural programming paradigm. However, the concepts can also be extended for other programming paradigms and different artefacts, including program documentation.

2.2.3. Software Product Quality Model of Kim and Lee

Kim and Lee [22] derived a model from the product quality model of the ISO/IEC 9126:2001. The authors determined the relative importance of the six high-level characteristics of the ISO standard from the point of view of the objectives of the project under examination. The order of the relative importance of the six characteristics was computed by applying the Analytic Hierarchy Process [34]. Those characteristics were kept for further investigation, the relative importance of which exceeded a defined threshold. In their case study they found three such attributes in the particular context: (1) reliability, (2) maintainability, and (3) portability.

They identified internal metrics for static code analysis and assigned these metrics to the three high-level characteristics of the ISO/IEC 9126:2001 model by considering the opinions of experts [22]. The metrics have directly been assigned to the high-level characteristics. Consequently, no intermediate level in the hierarchy with sub-characteristics was defined, i.e. the three characteristics formed categories rather than hierarchies.

The authors also presented the evaluation of a software component to illustrate the use of their model [22]. The critical places for improvement were identified in the component analysed. After performing amendments of the identified quality defects, the evaluation was carried out again, which verified the impact of the corrections.

2.2.4. Software Product Quality Model of the ISO/IEC 25010 Standard

The ISO/IEC 25000 standard family supersedes the ISO/IEC 9126 and ISO/IEC 14598 standard families. ISO/IEC 25010:2011 [13] defines a new quality in use model and a new software product quality model combining the internal and external models of the ISO/IEC 9126 standard family. However, it keeps the concepts laid down by the previous ISO/IEC 9126-1:2001 standard [13].

Terminology:

- Definition of internal, external view of quality and quality in use are taken over from the predecessor ISO/IEC 9126-1:2001 standard [14] but the internal and external software product quality models were combined to one software product quality model.
- Quality Measure Element (QME): measurable property of quality defined in ISO/IEC 25021:2007 [16].
- Quality Measure (QM): quality measure elements and a measurement function to calculate with. It is similar to the term metric in the ISO/IEC 9126-1:2001 standard. Initial list of quality measures was taken over from ISO/IEC TR 9126-2:2003 [17], ISO/IEC TR 9126-3:2003 [18] and ISO/IEC TR 9126-4:2004 [19].
- Quality attribute: low-level quality property, in contrast to the ISO/IEC 14598 standard family where the term attribute is used for the high-level quality properties of the ISO/IEC 9126 family.

Concepts:

The software product quality model introduces slight changes in the naming of the six high-level characteristics of the ISO/IEC 9126-1:2001 standard and adds two further high-level characteristics to the previous model: security, compatibility. The whole list of high-level quality characteristics: (1) functional suitability, (2) performance efficiency, (3) compatibility, (4) usability, (5) reliability, (6) security, (7) maintainability, (8) portability. In addition, the sub-characteristics were partially modified.

The quality in use model defines one additional high-level characteristic to the previous characteristics defined in ISO/IEC 9126-1:2001 and performs slight changes in the naming. The present high-level characteristics of the quality in use model: (1) effectiveness, (2) efficiency, (3) satisfaction, (3) freedom from risk, (4) context coverage. The new model also defined sub-characteristics which were not part of the previous quality in use model.

The new model description emphasizes the necessity of tailoring the model to the specific objectives of projects.

The new members of the ISO/IEC 25000 standard family: ISO/IEC 25022, 25023, 25024 are expected to be issued in the future. These standards will suspend the validity of the previous technical reports that define internal, external and quality in use metrics: ISO/IEC TR 9126-2, 9126-3, 9126-4.

These models are language-independent and independent of programming paradigms.

3 Towards Extending Present Quality Frameworks

The software product quality frameworks presented in the previous section show two basic approaches for describing software product quality: (1) the hierarchic approach depicted by the ISO/IEC 9126 and ISO/IEC 25000 standard families which have their roots in the early models; and (2) the non-hierarchic approach described by Dromey. The other frameworks tailor the first approach or its predecessors to the specific context of use. All the frameworks presented are the result of empirical research, which offers possibilities for changes and tailoring.

For this reason we will investigate the extensibility of the ISO/IEC 9126, ISO/IEC 25010 quality frameworks and the framework defined by Dromey.

This investigation includes (1) where the description of execution tracing quality could be placed in the existing models and (2) what methods the complete frameworks offer to describe execution tracing quality including the reflection of subjective uncertainty. Nevertheless, the property illustrating execution tracing quality also needs to be able to express the quality of execution tracing as a standalone model without the frameworks presented.

3.1. ISO/IEC 9126 Framework

The standard allows adaptations of the software product quality model defined in the scope of the ISO/IEC 9126 standard family. The model definition in ISO/IEC 9126-1:2001 is superseded by ISO/IEC 25010:2011 but the model and the quality metrics are not superseded until ISO/IEC 25022 and 25023 are issued. Therefore we include this model in our investigation.

Following the concepts and terminology of the present software product quality framework the following steps are necessary for extension:

- Defining which characteristics and sub-characteristics can locate the execution tracing related quality description.
- Defining one or more internal and external metrics related to the quality property of execution tracing. The internal and external metrics have to correlate and the internal metrics need to have predictive validity towards the external metrics.

Extension Method

Execution tracing quality significantly influences the effort needed for error analysis. This identifies by its nature a property which belongs to maintainability or any of its sub-characteristics.

The high-level characteristic maintainability comprises of five sub-characteristics: (1) analysability, (2) changeability, (3) stability, (4) testability, (5) maintainability compliance. With regard to the goal of execution tracing the sub-characteristic analysability offers a logical point to link to because it encompasses all metrics which describe how the software or its behaviour can be analysed.

After finding the location in the hierarchy, the metrics need to be defined. As the description of execution tracing quality needs to be able to describe the quality of execution tracing as a standalone model, it is not recommended to define more metrics because it would create a dependency on the ISO product quality framework. If a new metric is introduced, the execution tracing quality model can easily be linked to the ISO framework without developing dependencies on it. For this reason we define an internal metric and an external metric keeping the naming conventions of the standard: (1) Internal Execution Tracing Capability Metric and (2) External Execution Tracing Capability Metric.

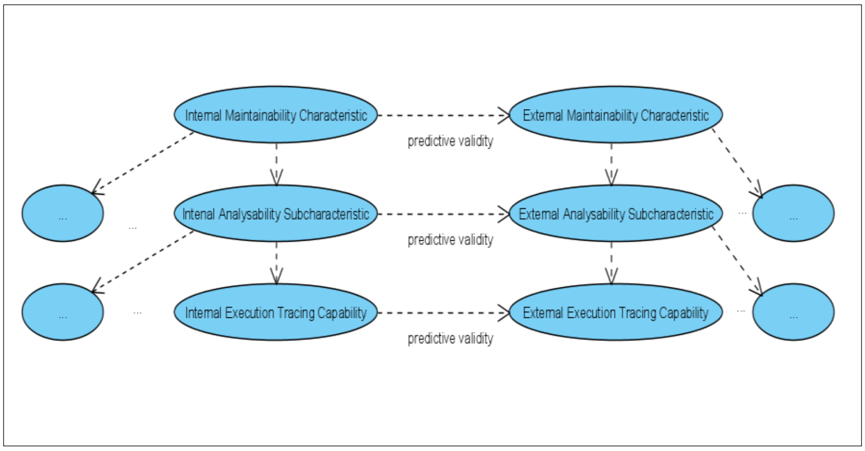
Internal Maintainability Characteristic
External Maintainability Characteristic
predictive validity
Intenal Analysability Subcharacteristic
External Analysability Subcharacteristic
predictive validity
Internal Execution Tracing Capability
External Execution Tracing Capability
predictive validity

Figure 1
Extending ISO/IEC 9126 with Execution Tracing Capability

The definition of the metric also requires identification of the inputs and the method how the metric can be calculated from these inputs. The inputs of the metrics are called quality measure elements according to the terminology of the standard ISO/IEC 25021:2007.

Benefits

The expected benefit of this extension is to consider execution tracing quality when the complete software product quality is assessed. In addition, the subjective uncertainty of the inputs i.e. the quality measure elements of the metrics, can also be reflected by the mathematical calculations, which can involve fuzzy logic.

No detrimental effects of this extension on the framework are known. The standard also encourages tailoring the software product quality model to specific needs of projects. Consequently, the extension is in accordance with the philosophy of the ISO/IEC 9126 standard family.

Existing Extension Results

Carvallo and Franch [4] point out that software evaluation is necessary from a technical point of view but their examination shows that non-technical factors related to licensing and supplier characteristics are even more important in case of commercial off-the-shelf (COTS) products. The authors propose to extend the ISO/IEC 9126 standard family to include non-technical factors in a uniform way. In the proposal the authors keep the hierarchical structure of the standard and define three high-level characteristics: (1) supplier, (2) costs, (3) product, which they decompose in fifteen sub-characteristics, which on the third level of the hierarchy are even further decomposed resulting in more than two hundred non-technical quality properties. They validated the extension of the model on different projects in the telecommunication industry on which they provide a brief summary in [4].

3.2. ISO/IEC 25010 Framework

The software product quality framework defined in ISO/IEC 25010:2011 revised the software product quality framework of ISO/IEC 9126-1:2001 as described before. The new standard kept the philosophy of the previous model. The changes in the model hierarchy do not affect the node analysability below maintainability. Thus, the extension point does not change in comparison to the ISO/IEC 9126-1 framework.

Nevertheless, combination of the internal and external software product quality models in ISO/IEC 25010:2011 needs to be considered. As ISO/IEC 25022 and 25023 are not issued to supersede ISO/IEC 9126-2:2003 and 9126-3:2003, the separation of internal and external quality views is also a viable option.

Extension Method

The extension possibilities described for ISO/IEC 9126-1:2001 can be used with the revised software product quality model this new standard introduced. The measures Internal Execution Tracing Capability and External Execution Tracing Capability were merged into a single Execution Tracing Capability measure to comply with the combined internal and external model and consequently it possesses both internal and external quality measure elements.

Special attention needs to be paid to the definition of the inputs of execution tracing quality and the description of the computation by which the quality of execution tracing can be computed. Definitions of new quality measures and quality measure elements are formalised and defined in the standard ISO/IEC 25021:2007.

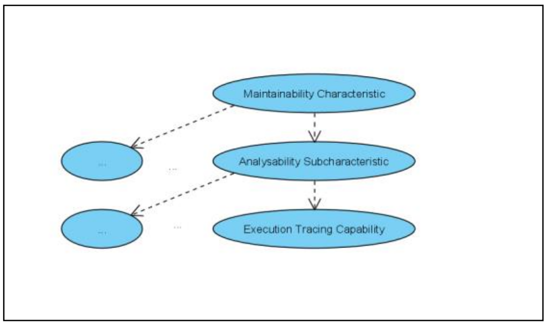
Maintainability Characteristic
...
Analysability Subcharacteristic
...
Execution Tracing Capability
...

Figure 2
Extending ISO/IEC 25010 with Execution Tracing Capability

Benefits

As with the previously discussed ISO/IEC 9126-1:2001 extension, the primarily expected benefit is to consider execution tracing quality when the complete software product quality is assessed. The subjective uncertainty of the quality measure elements of the defined quality measures can also be reflected by mathematical calculations including fuzzy logic.

In comparison to extending the ISO/IEC 9126-1:2001 framework, a further advantage is to reduce the dependency on the ISO-framework to one measure which needs to be linked to it, thus supporting the standalone application of the quality model describing execution tracing.

No detrimental effects of this extension on the framework are known. The standard also declares that tailoring the software product quality model of ISO/IEC 25010 to specific needs of projects is a must i.e. tailoring is more emphasised in the revised standard than in its predecessor. Consequently, the extension is in accordance with the philosophy of the standards.

Existing Extension Results

No extension attempts of ISO/IEC 25010 were found in the literature.

3.3. Dromey's Framework

For each section the terminology of the model in question is used. Dromey's model applies the word attribute in a different way to the ISO/IEC 9126 and ISO/IEC 25000 standard families.

Dromey handles three primary sets of entities in his framework without introducing hierarchies. The relationships of these sets depict the quality requirements and the criteria for assessment. The set of high-level quality attributes contains maintainability which definitely illustrates the category to which execution tracing quality needs to be assigned. Therefore the set of high-level quality attributes need to undergo no changes. Consequently, extension possibilities for sets of quality-carrying properties and structural forms need to be examined.

Extension Method

Because all the structural forms define programming language-level constructs in the original description, higher-level structural forms are also necessary to include entities on component-level or application-level. New quality-carrying properties need to be introduced in the framework to describe the input variables of execution tracing in a binary manner to show whether the property is present in the application under investigation or not.

Execution tracing related quality-carrying properties can be linked to the new structural forms and to the high-level attribute maintainability in order to establish relationships. Then following the bottom-up approach introduced in the model description, the optimal relationships for each structural form need to be defined which guarantee the good quality; moreover, to support the top-down approach the optimal relationships need to be defined between the quality attributes and the quality-carrying properties. These profiles give a measure that can be compared to the actual software under investigation to diagnose quality defects or to set quality targets.

The original definition of the framework only considers the procedural programming paradigm, which has to be kept otherwise the present model needs to be reworked significantly to create new quality-carrying properties. The model's basic principles also facilitate the accommodation to other programming paradigms with the introduction of new quality-carrying properties and structural forms to define new relationships.

Benefits

As with the ISO/IEC 9126-1:2001 extension, the primarily expected benefit is to consider execution tracing quality when the complete software product quality is assessed.

As detrimental effect the high number of new relationships between the necessary quality-carrying properties and structural form needs to be mentioned, if not only programming language level assessment is necessary. On the other hand, the model supports the procedural programming paradigm only, so extension to further programming paradigms would implicate additional quality-carrying properties which would result in additional relationships between the quality attributes, structural forms and quality-carrying properties. The high number of possible relationships which should be processed during quality assessment can make the model unmanageable.

Existing Extension Results

No extension attempts of Dromey's framework were found in the literature.

4 Discussion

The frameworks investigated in the previous section allow extensions to include execution tracing quality but their implementations differ significantly.

Dromey's model only describes code-level constructs and their quality considering the procedural programming paradigm. The principle of the model, however, can also be applied for higher-level constructs and additional programming paradigms. From the point of view of execution tracing, procedural programming does not cause difficulties although the usability of the model would significantly be reduced if no other programming paradigms could be represented. To encompass additional programming paradigms and higher-level artefacts, Dromey’s model requires considerable amounts of new quality-carrying properties and new structural forms. The high number of elements in both sets enhances the number of combinations through which relationships need to be expressed. Consequently, the execution tracing quality can be described at the cost of introducing more complexity in the model. In addition, the direct assignment of binary quality-carrying properties to high-level attributes leaves no room to uncertainty computations.

In contrast, ISO/IEC 9126 and 25010 offer an extension possibility and a sub-characteristic to which the description of execution tracing can be linked: maintainability and its analysability sub-characteristic. Linking is simple and requires considerably less effort than incorporating the illustrated changes in Dromey's model. Moreover, the quality measure or metric definitions complying with the standards allow the use of mathematical functions, by which subjective uncertainty computation can also be implemented.

If execution tracing quality were to be described by means of Dromey’s framework, then it could not be used as an independent model because the framework requires a specific implementation. On the contrary, linking the description of execution tracing quality to the ISO/IEC software product quality frameworks facilitates its existence as an independent model.

ISO/IEC product quality models are more widespread than Dromey’s model and they are known to a larger audience, as evidenced by the high number of publications relating to these standards, moreover by the models based on the ISO/IEC framework presented in the previous chapter. In addition, execution tracing quality can be encompassed with significantly less effort in the ISO/IEC standards than in Dromey’s model.

Conclusion

Execution tracing is an important property that needs to be considered in quality frameworks to truly reflect the overall view of software product quality. Dromey's model allows extensions to include execution tracing quality although it requires significant changes in the present model. The model's philosophy does not support mathematical operations on quality-carrying properties and, therefore implementing subjective uncertainty computations is infeasible at present.

Software product quality frameworks of the ISO/IEC standards allow extensions and have a defined method to do so. Moreover, they also offer a natural linking point for execution tracing quality with the analysability sub-characteristic of maintainability. They can also allow mathematical computations that make the implementation of subjective uncertainty computations possible.

In conclusion, execution tracing quality should be linked to the software product quality framework of the ISO/IEC 25010 standard with the observation of the rules for defining quality measures and quality measure elements. This would facilitate the consideration of execution tracing quality when the whole software product quality is assessed; furthermore, it would ensure a framework for incorporating the impacts of subjective uncertainty resulting from the quality measurement process.

In summary, the findings of the paper include: (1) execution tracing quality should be reflected by the software product quality frameworks, (2) the current software product quality frameworks offer the possibility for extension but with significantly different efforts, (3) the ISO/IEC software product quality frameworks facilitate mathematical computations for defining measures or metrics, which allow to capture and implement subjective uncertainty and (4) metrics and a measure for execution tracing quality are outlined for the ISO/IEC 9126-1 and the ISO/IEC 25010 software product quality frameworks.

References

[1] Banker, R. D., and S. Slaughter. “A Field Study of Scale Economies in Software Maintenance.” *Management Science* 43, No. 12 (1997): 1709-1725

[2] Boehm, B. W., J. R. Brown, and M. Lipow. “Quantitative Evaluation of Software Quality.” *Proceedings of the 2nd International Conference on Software Engineering.* 1976

[3] Buch, I., and R. Park. “Improve Debugging and Performance Tuning with ETW.” *MSDN Magazine, [Online], [Accessed: 01.01.2012], Avaliable from: http://msdn.microsoft.com/en-us/magazine/cc163437.aspx,* 2007

[4] Carvallo, J., and X. Franch. “Extending the ISO/IEC 9126-1 Quality Model with Non-Technical Factors for COTS Components Selection.” *Proceedings of the 2006 International Workshop on Software Quality.* 2006. 9-14

[5] Csallner, C., and Y. Smaragdakis. “DSD-Crasher: A Hybrid Analysis Tool for Bug Finding.” *ACM Transactions on Software Engineering and Methodology (TOSEM).* 2008

[6] Csernyak, L., G. Horvath, J. Horvath, S. Lorincz, S Molnar, and A Stern. *Matematika a Kozgazdasagi Alapkepzes Szamara (Translated Title: Mathematics for the Bachelor Curricula in Economics, Probability Theory).* Budapest: Nemzeti Tankonyvkiado, 2007

[7] Dromey, R. "A Model for Software Product Quality." *IEEE Transactions on Software Engineering.* 1995, 146-162

[8] Ernst, M. D. "Static and Dynamic Analysis: Synergy and Duality." *In Proceedings ICSE Workshop on Dynamic Analysis*, 2003: 24-27

[9] Galli, Tamas, Francisco Chiclana, Jenny Carter, and Helge Janicke. "Modelling Execution Tracing Quality by Means of Type-1 Fuzzy Logic." *Acta Polytechnica Hungarica*, Vol. 10, No. 8, 2013: 49-67

[10] Godefroid, P., and N. Nagappan. *Concurrency at Microsoft – An Exploratory Survey, [Online], 2007, [Accessed: 24.01.2012.], Available from: http://research.microsoft.com/en-us/um/people/pg/public_psfiles/ec2.pdf.* Microsoft Research

[11] Hovemeyer, D., and W. Pugh. "Finding Bugs Is Easy." *OOPSLA: Object-Oriented Programming, Systems, Languages & Applications, [Online], [Accessed: 14.02.2012], Available from: http://www.cs.nyu.edu/~lharris/papers/findbugsPaper.pdf.* 2004

[12] IBM. "Understanding Execution Traces, [Online], [Accessed: 05.02.2013], Available from: http://pic.dhe.ibm.com/infocenter/brdotnet/v7r1/index.jsp?topic=%2Fcom.ibm.websphere.ilog.brdotnet.doc%2FContent%2FBusiness_Rules%2FDocumentation%2F_pubskel%2FRules_for_DotN."

[13] International Organization for Sandardization. "ISO/IEC 25010:2011, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models." 2011

[14] International Organization for Sandardization. "ISO/IEC 9126-1:2001, Software engineering -- Product quality -- Part 1: Quality model." 2001

[15] International Organization for Standardization. "ISO/IEC 14598:1999, Information technology -- Software product evaluation -- Part 1: General overview." 1999

[16] International Organization for Standardization. "ISO/IEC 25021:2007, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Quality measure elements." 2007

[17] International Organization for Standardization. "ISO/IEC TR 9126-2:2003, Software engineering -- Product quality -- Part 2: External metrics." 2003

[18] International Organization for Standardization. "ISO/IEC TR 9126-3:2003, ." *Software engineering -- Product quality -- Part 3: Internal metrics*, 2003

[19] International Organization for Standardization. "ISO/IEC TR 9126-4:2004, Software engineering -- Product quality -- Part 4: Quality in use metrics." 2004

[20] Kahn, A. *Simulation of Message Passing Programs, [Online] http://may.cs.ucla.edu/projects/sesame/publications/sundeep_diss_html/node43.html, University of California.* 1997

[21] Karahasanovic, A., and R. Thomas. "Difficulties Experienced by Students in Maintaining Object-oriented Systems: An Empirical Study." *Proceedings of the 9th Australasian Conference on Computing Education*, 2007: 81-87

[22] Kim, C., and K. Lee. "Software Quality Model for Consumer Electronics Product." *Proceedings of the 9th International Conference on Quality Software.* 2008, 390-395

[23] Kitchenham, B., and S. Pfleeger. "Software Quality: the Elusive Target." (IEEE Software) 13, No. 1 (1996): 12-21

[24] Klir, G. J., U. H. St.Clair, and B. Yuan. *Fuzzy Set Theory Fundations and Applications.* Prentice Hall Ptr, 1997

[25] Koskinen, J. *Software Maintenance Costs.* [Online], 2010, [Accessed: 23.01.2012], Available from: http://users.jyu.fi/~koskinen/smcosts.htm

[26] Krishnan, M. S., T. Mukhopadhyay, and C. H. Kriebel. "A Decision Model for Software Maintenance." *Information Systems Research* 15, No. 4 (2004): 396-412

[27] Laddad, R. *AspectJ in Action.* Manning, MEAP, Second Edition, 2009

[28] Martin, A. "Debugger, Real-Time Trace, Logic Analyser, Long-Term Trace ETMv3, [Online], [Accessed: 05.02.2013], Available from: http://www.lauterbach.com/publications/long_term_trace_etmv3.pdf, Lauterbach GmbH." 2009

[29] McCall, J. A., P. K. Richards, and G. F. Walters. *Factors in Software Quality, Concept and Definitions of Software Quality.* Technical Report, RADC-TR-77-369, Rome Air Development Center, Air Force System Command, Griffis Air Force Base, New York 13441, [Online], [Accessed: 21.10.2011], Available from: http://handle.dtic.mil/100.2/ADA049014, 1977

[30] Microsoft Co. "Tracing WMI Activity (Windows), [Online], [Accessed: 05.02.2013], Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/aa826686%28v=vs.85%29.aspx." 2012

[31] Panda, D., R. Rahman, and D. Lane. *EJB 3 in Action.* Manning Publications Co., 2007

[32] Qu, D., A. Roychoudhury, Z. Lang, and K. Vaswani. *Darwin: An Approach for Debugging Evolving Programs.* Microsoft Research, [Online], 2009, [Accessed: 24.01.2012], Availavle from: http://research.microsoft.com/apps/pubs/default.aspx?id=80898

[33] Research Triangle Institute. *RTI Project Number 7007.011, The Economic Impacts of Inadequate Infrastructure for Software Testing.* National Institute of Standards and Technology, U.S Department of Commerce, Technology Administration, National Institute of Standards and Technology, U.S Department of Commerce, Technology Administration, [Online], 2002, [Accessed: 20.01.2012], Avaliable from: http://www.nist.gov/director/planning/upload/report02-3.pdf, 2002

[34] Saaty, T. L. *The Analytic Hierarchy Process.* New York: McGraw-Hill, 1980

[35] SAP. "Using the Technical Trace and Log, [Online], [Accessed: 05.02.2013], Available from: http://help.sap.com/saphelp_nwpi71/helpdata/en/3a/63e540aa827e7fe10000000a1550b0/content.htm."

[36] Shi, Z. "Visualizing Execution Traces, Master Thesis." *[Online], [Accessed: 17.05.2011], Available from: http://www.mcs.vuw.ac.nz/comp/graduates/archives/mcompsc/reports/2004/Zhenyu-Shi-final-report.pdf*, 2005

[37] Spillner, A., T. Linz, and H. Schaefer. *Software Testing Foundations.* Santa Barbara, CA: Rockey Nook Inc., 2007

[38] Spinczyk, O., D. Lehmann, and M. Urban. "AspectC++: an AOP Extension for C++." *Software Developers Journal*, 2005: 68-74

[39] Uzelac, V., A. Milenkovic, M. Burtscher, and M. Milenkovic. "Real-time Unobtrusive Program Execution Trace Compression Using Branch Predictor Events." *CASES 2010 Proceedings of the 2010 international conference on Compilers, Architectures and Synthesis for Embedded Systems, ISBN: 978-1-60558-903-9*, 2010

[40] Winterfeldt, D. *Spring by Example.* Spring Tutorial, [Online], [Accessed: 19.12.2012], Available from: http://www.springbyexample.org/examples/aspectj-ltw-spring-config.html

[41] Young, M. "Symbiosis of Static Analysis and Program Testing." *In Proc. 6th International Conference on Fundamental Approaches to Software Engineering*, 2003: 1-5