J. Inf. Technol. Appl. Manag. 23(3): 71~86, September 2016
http://dx.doi.org/10.21219/jitam.2016.23.3.071
ISSN 1598-6284 (Print)
ISSN 2508-1209 (Online)

Cause-and-Effect Perspective on Software Quality : Application to ISO/IEC 25000 Series SQuaRE's Product Quality Model

Seokha Koh*

Abstract

This paper proposes a new software quality model composed of a hierarchy of software quality views and three software quality characteristics models. The software view hierarchy is composed of two levels : end view and means view at the first level, contingency view and intrinsic view as sub-views of means view. Three software quality characteristics models are activity quality characteristics model, contingency quality characteristics model, and intrinsic quality characteristics model, which correspond to end view, contingency view, and intrinsic view respectively.

This paper also reclassifies characteristics of ISO/IEC 25000 series SQuaRE's software product quality model according to the proposed software quality model. The results illustrate clearly the shortcomings of SQuaRE's product quality model and how to overcome them. First of all, most of SQuaRE's product characteristics should be redefined and conceptually clarified according to the views on which they are really rested. Much more characteristics should be supplemented too. After that, rigorous empirical researches will become relevant. Causal relationships between activity quality characteristics and characteristics of means view should be empirically researched.

Keywords : Software Quality, ISO/IEC 25000 Series, SQuaRE(Systems and software Quality Requirement and Evaluation), Software Quality View, Software Quality Model, Cause and Effect Relationship

Received : 2016. 05. 25.   Revised : 2016. 06. 29.   Final Acceptance : 2016. 09. 19.
※ This work was supported by the research grant of Chungbuk National University in 2014.
* Department of Management Information Systems, Chungbuk National University, 1 Chungdae-ro Seowon-gu Cheongju, Chungbuk 362-763, Korea, Tel : +82-43-261-7062, Fax : +82-43-273-2355, e-mail : shkoh@cbnu.ac.kr

1. Introduction

This paper proposes a new software quality model and reclassifies characteristics of ISO/IEC (The International Standard Organization/The International Electrotechnical Commission) *25000* Series SQuaRE's (System and software product Quality Requirements and Evaluation) software product quality model according to the proposed model. The results illustrate clearly the shortcomings of SQuaRE's product quality model and how to overcome them.

ISO/IEC issued ISO/IEC 25000 in 2005, ISO/IEC 25020 and 25030 in 2007, ISO/IEC 25010 and 25040 in 2011. ISO/IEC also published many other documents regarding SQuaRE. SQuaRE replaces ISO/IEC 9126 issued in 1991 and revised during 2001~2004. As well as ISO/IEC 9126, however, SQuaRE still suffers from ambiguity, inconsistency, and contradictions, and is un-suitable to measure the design quality of software product [Al-Kilidar, 2005; Haboush et al., 2014; Kitchenham and Pfleeger, 1996; Koh and Whang, 2016].

SQuaRE defines the quality of a software product or a system as '*the degree to which it satisfies the stated and implied needs of its various stakeholders, and thus provides value*'[1] [ISO/IEC 25022.2 : 2015]. This definition implies that the quality of the product may vary as stakeholders or their needs regarding a software product vary. That is, SQuaRE defines software quality as something that is neither invariant nor intrinsic.

SQuaRE defines three views : internal, external, and in-use. According to SQuaRE, *internal software quality is mainly related to static properties of the software and has an impact on external software quality, which again has an impact on quality in use* [IOC/IEC 25030:2007]. SQuaRE, however, defines only two software quality models : quality in use model which is rest on in-use view and product quality model which is rest on both internal view and external views simultaneously. This implies that SQuaRE may not treat cause and effect relationships properly, in fact. In this paper, this issue and associated issues will be addressed in depth. The results will provide indispensable clues to resolve SQuaRE's shortcomings and to construct a more integrative and consistent software quality model.

From now on in this paper, the issues regarding only software quality will be addressed. The issues regarding the quality of system will be excluded from discussions unless they are related with the quality of software. For example, security is regarded as a system-level characteristic in this paper. Security of a system depends on chiefly security software and procedures rather than individual application software products : *As well as data stored in or by a product or system, security also applies to data in transmission* [ISO/IEC 25010 : 2011, p. 14]. So, security and its sub-characteristics will be excluded from further discussion. Without any further notice, the term quality will be used to denote the quality of software products.

1) In this paper, italic font denotes that corresponding part is quoted with no or only slight changes from the cited literature.

<Table 1> Types of Software Activity Cited in the Product Quality Characteristic Definitions

Characteristic Sub-characteristic	Type of Activity	Role of Entity Performing the Activity
Usability	Using	End user
Accessibility	Accessing	
Appropriateness recognizability	Recognizing appropriateness	
Learnability	Learning how to use	
Operability	Operating/controlling	
Portability	Transferring	System operator
Installability	Installing/uninstalling	
Adaptability	Adapting	Maintainer
Maintainability	Modifying	
Analyzability	Analyzing	
Modifiability	Modifying	
Testability	Testing	
Reusability	Reusing an asset	Developer of other software products

Source : Koh and Whang [2016].[3)]

2. Dependency in Definitions of Product Quality Characteristics

SQuaRE's product quality model has 8 product quality characteristics (in contrast to ISO 9126's 6), and 31 sub-characteristics (in contrast to ISO 9126's 21[2)]). Among 8 product quality characteristics, usability, portability, and maintainability are directly related with software activities such as using, transferring, and modifying, respectively (refer <Table 1> and underlined parts of <Appendix>). Koh and Whang [2016] call such characteristics activity quality characteristics. In this paper, the software activity is defined as the activity which is performed on a software product by external entities other than the product itself. The operation which is performed by the target software itself is not classified as software activity.

According to this definition, 'providing functions' in functional suitability is not software activity since the entity that 'provides functions' is the target product itself. By the same token, exchanging in compatibility, using in performance efficiency, recovering and re-establishing in recoverability, and performing in reliability, are not regarded as software activities. So, corresponding characteristics are not classified as activity quality characteristics.

SQuaRE explains replaceability as a composite concept of installability and adaptability [ISO/

2) Compliance with laws and regulations is regarded as a sub-characteristic, as usual. Compliance is included as a sub-characteristic of each and every characteristic. So, compliance to each characteristic may be regarded as a sub-characteristic, increasing the number of sub-characteristics to 26. Or compliance may be regard as a characteristic which has 6 sub-characteristics, increasing the numbers of characteristic and sub-characteristic to 7 and 26, respectively. ISO/IEC 25010 regards compliance as a part of the overall requirements, rather than as a specific part of software quality.

3) Availability, recoverability, and replaceability are deleted from the original table.

IEC 25010 : 2011, p. 16]. Then, it will be redundant with the characteristics. Moreover, SQuaRE explains that it is not a characteristic of the original product but that of an already adapted or modified version of the current target product: it refers whether or not *an already adapted or modified version of the current target product can be used in place of the present one* [ISO/IEC 25010 : 2011, p. 16]. That is, SQuaRE does not regard it as a generic quality characteristic. Moreover, it is a composite concept of installability, compatibility, functional suitability, performance efficiency, or etc. according to the explanation. It will be excluded from further discussions in this paper too.

Although most sub-characteristics of usability, portability and maintainability are explicitly defined to involve software activity, however, some sub-characteristics of them are not related with software activity. For example, among sub-characteristics of usability, user error protection and user interface aesthetics are not related with any software activity. Although user error protecting involves 'protecting users', the entity that protect users is defined to be the target product. So, user error protection is not classified as an activity quality characteristic. Among sub-characteristics of maintainability, it is obvious that modularity is not related with software activity.

It is noticeable that accessing is involved in the definitions of both accessibility and availability. In this paper, availability is presumed not to be related with individual accessing activities by end users, but to be related with the state in which the product can be accessed.4) Otherwise, it will be redundant with accessibility. In this regard, availability is affected by the target product and the target system, but affected by neither environments nor contexts.

Modifying also appears in the definitions of both maintainability and its sub-characteristic modifiability. The definitions of these characteristics are virtually the same, although their expressions are slightly different. The only virtual difference is that '*modifying without introducing defects or degrading existing product quality*' is added in modifiability. Then, is it alright for 'defects or degrading existing product quality' to be introduced in maintaining? The definition of these two characteristics should be differentiated for these characteristics not to be redundant.

SQuaRE classifies adaptability as a sub-characteristic of portability. At the same time, SQuaRE also regards adapting as a sub-type of modification or an activity alternative to modification : *Modifications can include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications: and both adaptations and modifications include those carried out by specialized support staff, and those carried out by business or operational staff, or end users* [ISO/IEC 25010 : 2011, pp. 14-15]. SQuaRE even regards installing as a sub-activity of maintaining : *Maintainability includes installation of updates and upgrades* [ISO/IEC 25010 : 2011, pp. 14-15]. According to SQuaRE's explanations, adaptability may not be classified as a sub-cha-

4) <Table 1> is identical with the corresponding table in Koh and Whang [2016] except that 'availability' is deleted.

racteristic of portability.

Adapting is typically regarded as a sub-type of maintaining or post life cycle changing [Abran and Nguyenkim, 1993; ANSI/IEEE Std. 729-1983; Dekleva, 1992; Glass, 1996; Hatton, 2007; Helms and Weiss, 1984; Herraiz et al., 2013; IEEE Std. 1219-1998; ISO/IEC 14764-2006; Kemerer and Slaughter, 1997; Koh and Han, 2015; Lients and Swanson, 1980; Sneed, 1996]. That is, an activity which is adapting is also maintaining.

Analyzing and testing are not sub-types, but are sub-activities of maintaining. Neither analyzing nor testing is maintaining by themselves. They are ones of various activities which constitute maintaining together. That is, to maintain a product, one should analyze, test the product, and even do something other.

On the other hand, modularity is neither a sub-type nor a sub-activity of maintaining. SQuaRE recognizes that both modularity and analyzability influence maintainability : *Modifiability is a combination of changeability and stability and can be influenced by modularity and analyzability* [ISO/IEC 25010 : 2011, p. 15]. It is right that both modularity and analyzability influence maintainability. This seems the reason that SQuaRE classifies both the characteristics as sub-characteristics of maintainability. The relationship between modularity and maintainability is cause-and-effect relationship, however, while that between analyzability and maintainability is correlational. Moreover, modularity influences analyzability too.

SQuaRE defines some product quality characteristics to be dependent on contexts, environments, or conditions, (refer <Table 2> and underlined parts of <Appendix>). It defines context of use and quality in use as '*users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used*' and '*the degree to which a product can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use*,' respectively [ISO/IEC 25010 : 2011, p. 18; ISO/IEC 25030 : 2007, p. 20]. It defines users to encompass various '*stakeholders who provide support such as maintainers, analyzers, porters, installers, content providers, system managers/administrators, and security managers as well as primary users who interact with the system to achieve the primary goals and indirect users who receive output, but does not interact with system*' [ISO/IEC 25010 : 2011, pp. 5-6]. It also defines the activity 'using' to encompass the various software activities that these stakeholders perform, for example, such as maintaining and porting.

SQuaRE emphasizes that '*each of these types of user has needs for quality in use and product quality in particular contexts of use*' [ISO/IEC 25010 : 2011, pp. 5-7]. In maintaining or porting, for example, maintainers have needs for effectiveness, efficiency, satisfaction, freedom from risk, reliability, security context coverage, learnability, and accessibility' [ISO/IEC 25010 : 2011, pp. 5-7]. SQuaRE also explains that *the term usability has a similar meaning to quality in use, but excludes freedom from risk and context coverage* [ISO/IEC 25022.2 : 2015]. These examples illustrate that SQuaRE regards every types of software activities is dependent on contexts.

〈Table 2〉 Dependency of SQuaRE's Product Quality Characteristics

Dependent on	Activity-Related	Not AR
Contexts	accessibility learnability USABILITY*	
Environments	adaptability (pairs of environments) installability PORTABILITY (pairs of environments)	COMPATIBILITY
Conditions		FUNCTIONAL SUITABILITY PERFORMANCE EFFICIENCY RELIABILITY
None	analyzability appropriateness recognizability-Usability** MAINTAINABILITY modifiability operability-Usability reusability testability	availability-Reliability fault tolerance-Reliability capacity-Performance efficiency coexistence-Compatibility functional appropriateness-Functional suit. functional completeness-Functional suitability functional correctness-Functional suitability interoperability-Compatibility maturity-Reliability modularity recoverability-Reliability resource utilization-Performance efficiency time behavior-Performance efficiency user error protection user interface aesthetics

* Upper case denotes that the characteristic is a super-characteristic.
** Pair of a sub-characteristic and its super-characteristic. For the pair, no dependency in the sub-characteristic is specified, but the author supposes, subjectively, that the dependency in the super-characteristic is inherited to the sub-characteristic.

SQuaRE, however, explicitly declares context dependency in definitions of only 3 characteristics usability and its two sub-characteristics accessibility and learnability. For adaptability, installability, and portability, SQuaRE explicitly declares dependency on only environments. For the other activity quality characteristics, SQuaRE does not explicitly declare any dependency in their definitions. But it is reasonable to interpret that the characteristics inherit the dependency of their super-characteristics. User error protection and user interface aesthetics, however, do not inherit context dependency form their super-characteristic usability since they do not involve any sub-activity of using.

SQuaRE declares functional suitability, performance efficiency, and reliability depend on conditions. It, however, does not specify what the term condition means : context, or environment, or something other than environments or contexts, or everything including environment and context.

In this paper, a characteristic of a software product that is not dependent on anything other than the product itself will be called an intrinsic characteristic. An intrinsic characteristic remains unique unless the product itself changes. A characteristic which depends on something other than the product, for example, such as system, contexts, environments, or conditions is not an intrinsic characteristic. That is, a characteristic which cannot be measured without referring something other than the target product itself is not intrinsic. Modularity is a typical example of intrinsic quality characteristic. It can be evaluated without referring something other than the software product itself.

All super-characteristics of product quality model are not intrinsic. Six super-characteristics are defined to be dependent on contexts, environments, or conditions. Although any dependency is not defined for maintainability, maintainability is an activity characteristic and depends on contexts.

3. Discussions

Regarding activity quality characteristics, SQuaRE proposes only the ends that the developer should pursuit without suggesting anything about how to realize the ends. Activity quality characteristics declare that software products should be good for software activities such as using, maintaining, and porting. But they, by themselves, do not provide any information about how to make software products good for such activities. They are even misleading.

Most of all, SQuaRE's product quality model does not properly classify types of software activity. For example, adapting is not a sub-type of porting although it is classified as sub-characteristic of portability.

Reusing is also not a sub-activity of maintaining according to its very definition : '*Reusing an asset in more than one system or in building other assets*' [ISO/IEC 25010 : 2011, p. 15] is not involved in maintaining the asset or the product containing the asset. Moreover, a software product as a whole is never used unchanged to develop another software product. It is an asset contained in the software product, which is reused in '*more than one system or in building other assets.*' If reusability refers 'the degree to which a product can be changed to be used in more than one system or in building other assets,' it is redundant with adaptability or modifiability. Reusability should be defined to be a characteristic of modules in a software product.

Another problem of the product quality model is that some super-characteristics include both activity characteristics and non-activity characteristics as their sub-characteristics simultaneously. For example, maintainability includes both modularity and analyzability. Modularity, however, may influence both maintainability and analyzability. This suggests that it may not be desirable to combine activity characteristics and non-activity characteristics into a model.

We define means view as an effort to find out the causes of something. Means view and end view are associated with cause and effect, respectively. Regarding software quality, means view represents the effort to find out the characteristics of software products that make the products good for various kinds of software activity. In this regard, all activity quality characteristics are of end view. Intrinsic view represents an effort to find out intrinsic characteristics which facilitate software activities. Contingency view represents an effort to find out non-intrinsic characteristics such that the degree to which they facilitate software activities varies according to contingencies. In this paper, the quality models rested on end view, contingency view, and intrinsic view will be called activity quality characteristics model, contingency quality characteristics model, and intrinsic quality characteristics model, respectively.

We presume following premise, proposition and hypotheses valid without rigorous validation or verification.

Premise : Views regarding software quality may be classified into end view and means view. Means view may be classified further into contingency view and intrinsic view.

Proposition : Quality characteristics of means view should be defined and measured in order to facilitate the effort to enhance quality characteristics of end view during development or maintenance of the software product.

Hypothesis 1 : Activity quality characteristics of SQuaRE's product quality model are of end view while the other product quality characteristics are of means view.

Hypothesis 2 : Modularity is the only intrinsic characteristic.

SQuaRE recognizes that causes and effects should be distinguished : *Internal software quality is mainly related to static properties of the software and has an impact on external software quality, which again has an impact on quality in use* [IOC/IEC 25030 : 2007]. If internal quality and external quality correspond to the cause and effect respectively, then a quality characteristic cannot be both internal and external simultaneously. SQuaRE, however, does not specify whether each product quality characteristic is internal or external.

SQuaRE does not distinguish a characteristic and the result of activity affected by the characteristic too. For example, SQuaRE explains that *maintainability can be interpreted as either an inherent capability of the product to facilitate maintenance activities, or the quality in use experienced by the maintainers for the goal of maintaining the product or system* [ISO/IEC 25010 : 2011, p. 14]. According to this explanation, by maintainability, SQuaRE refers simultaneously both the 'capability' of a software product that affect both maintaining and quality in use as the result of maintaining the product. However, the one as an intrinsic characteristic of the product influences the latter indirectly by influencing maintaining the product. It is not desirable to let the cause and effect have the same name.

In fact, SQuaRE regards evaluating product quality externally as estimating quality in use : *Quality in use can be assessed by observing representative users carrying out representative tasks in a realistic context of use; its measurement may be obtained by simulating a realistic working environment (for instance in a usability laboratory) or by observing operational use of the product; and some external usability measures are tested in a similar way, but evaluate the use of particular product features dur-*

ing more general use of the product to achieve a typical task as part of a test of the quality in use [ISO/IEC 25022.2 : 2015]. Apparently, an estimate of something cannot be its cause.

Including activity quality characteristics in a quality model has no effect other than declaring that a software product should be good for the activities associated with them. How much a product will be good for such activity should be estimated or forecasted during development or maintenance of the product. The activity quality characteristics, however, tell nothing about how to make a software product good for software activities. So, they are of end view.

The answer should be given by the means view. For example, maintainability may be enhanced by enhancing modularity. Usability seems to be enhanced by enhancing functionality suitability, performance efficiency, reliability, and/or all of their sub-characteristics. Usability can be enhanced by enhancing its two non-activity sub-characteristics too. Installability seems to be enhanced by enhancing compatibility. <Table 3> shows the author's subjective opinion on the cause and effect relationships between contingency quality characteristics and activity quality characteristics. <Table 3> also shows the author's subjective opinion on the dependency of contingency quality characteristics. The contents of <Table 3> should be elaborated much more.

Modularity is the typical example of intrinsic quality characteristic. It is obvious that modularity cannot be measured externally. Whether user error protection is intrinsic or not depends on how user error is defined. If user error is defined to depend on external factors outside the target product, then it cannot be intrinsic. If user error is defined not to depend on external factors, however, then the usefulness of user error

〈Table 3〉 Influence and Dependency of Contingency Quality Characteristics : A Subjective Ppinion

Characteristic	Sub-characteristic	Influences	Depends on
Compatibility	Co-existence	Installing	System (software portfolio)
	Interoperability	Installing	System (software portfolio)
Functional suitability	Functional appropriateness	Using	Social and organizational environment (work and task)
	Functional completeness	Using	Social and organizational environment (work and task)
	Functional correctness	Using	Social and organizational environment (work and task)
Performance efficiency	Capacity	Using	System (data volume and structure)
	Resource utilization	Using	System (data volume and structure)
	Time behavior	Using	System (data volume and structure)
Reliability	Availability	Using	System
	Fault tolerance	Using	System
	Maturity	Using	System
	Recoverability	Using	System
Usability	User error protection	Using	System, environment (work and task, physical)
	User interface aesthetics	Using	System (interface device), physical environment (illumination, noise, etc.)

protection as a quality characteristic seems to diminish. So, it seems better to define user error to depend on contexts.

4. Conclusions

This paper proposes a new software quality model composed of a hierarchy of software quality views and three software quality characteristics models. The software view hierarchy is composed of two levels : end view and means view at the first level, contingency view and intrinsic view as sub-views of means view. Three software quality characteristics models are activity quality characteristics model, contingency quality characteristics model, and intrinsic quality characteristics model, which correspond to end view, contingency view, and intrinsic view respectively. The intrinsic quality characteristic is defined as the quality characteristic that is not dependent on anything other than the product itself.

In this paper, security, replaceability, and reusability are excluded from discussions about software product quality. Security is regarded as a quality characteristic of a system. That is, the portion of influence that individual applications exert on security of a system is regarded negligible. Replaceability is regarded as a composite characteristic of installability and compatibility. Reusability is regarded a characteristics of a module or component rather than that of a software product.

This paper reclassifies ISO/IEC 25000 Series SQuaRE's remaining software product quality characteristics according to the proposed software quality model : usability along with its sub-characteristics excluding user error protection and user interface aesthetics, maintainability along with its sub-characteristics excluding modularity, and portability along with its sub-characteristics installability and adaptability into activity quality; modularity into intrinsic quality; and the other characteristics into contingency quality.

Activity quality characteristics declare what the software product should good for. A software product should good for various software activities such as using, maintaining, and porting. So, software activities constitute the end regarding software quality. The types of software activity in SQuaRE's product quality model, however, are not sufficiently comprehensive. Moreover, the classification hierarchy is even misleading. A software activity classification hierarchy which is exhaustive and mutually exclusive should be developed. The classification criteria should be clearly specified too.

The quality model of means view as a whole should be defined to facilitate enhancing activity quality. Each quality characteristic of end view should be enhanced by enhancing some quality characteristic of means view. If not, the model is not sufficiently comprehensive. At the same time, some quality characteristic should be enhanced by enhancing a quality characteristic of means view. If not, such a quality characteristic of means view is of little use.

During development or maintenance of a software product, the developer or maintainer should be able to enhance quality characteristics of means view to enhance activity quality of the product. A valid and reliable measurement of activity quality is frequently unavailable. Especially when an executable version is unavailable, a valid and reliable measurement of usability cannot be obtained. So, it should be possible to measure quality characteristics of means view validly and reliably even when an executable version is unavailable.

Except modularity, all characteristics of SQuaRE's product quality model are defined to depend on contexts, environment, or conditions. It is not sure, however, what context, environments, and conditions mean. They should be redefined more specifically to get rid of such vagueness.

The quality of software is one of the most fundamental issues of software : It means almost everything except price of software products. Without the proper consideration for software quality, the software industry cannot develop properly. Especially, purchasers and users of software cannot be protected properly without a proper consideration for software quality. The academy should contribute to protect them by establishing proper software quality models and by influencing the software industry to increase the quality of software products that it produces.

The proposed software quality model composed of the characteristics discussed in this paper is not sufficiently comprehensive. Much more characteristics should be supplemented. Moreover, most of SQuaRE's product quality characteristics should be redefined and conceptually clarified according to the views on which they are rested. Only after that, rigorous empirical researches will be relevant. Causal relationships between activity quality characteristics and characteristics on means view should be empirically researched.

References

[1] Abran, A. and Nguyenkim, H., "Measurement of the Maintenance Process from a Demand-Based Perspective", *Journal of Software Maintenance : Research and Practice*, Vol. 5, No. 2, 1993, pp. 63-90.

[2] Al-Kilidar, H., Cox, K., and Kitchenham, B., "The Use and Usefulness of the ISO/IEC 9126 Quality Standard", *Proceedings of International Symposium on Empirical Software Engineering 2005*, IEEE, 2005, pp. 126-132.

[3] ANSI/IEEE Std. 729-1983, *IEEE Standard Glossary for Software Engineering Terminology*, 1983.

[4] Dekleva, S. M., "Software Maintenance : 1990 Status", *Journal of Software Maintenance : Research and Practice*, Vol. 4, No. 4, 1992, pp. 233-247.

[5] Glass, R. L., "Results of the First IS State-of-the-Practice Survey", *The Software Practitioner*, 1996, pp. 3-4.

[6] Haboush, A., Alnabhan, M., AL-Badareen, A., Al-Nawayseh, M., and EL-Zagmouri, B., "Investigating Software Maintainability Development : A Case for ISO 9126", *International J. of Computer Science Issues* (IJCSI), Vol. 11, No. 2, 2014, pp. 18-23.

[7] Hatton, L., "How Accurately Do Engineers Predict Software Maintenance Tasks?", *Computer*, 2007, pp. 64-69.

[8] Helms, G. L. and Weiss, I. R., "Applications

Software Maintenance : Can It Be Controlled?", *ACM SIGMIS Database*, Vol. 16, No. 2, 1984, pp. 16-18.

[9] Herraiz, I., Rodriguez, D., Robles, G., and Gonzalez-Barahona, J. M., "The Evolution of the Laws of Software Evolution : A Discussion Based on a Systematic Literature Review", *ACM Computing Surveys*, Vol. 46, No. 2, 2013.

[10] IEEE, *IEEE Std. 1219-1998, IEEE Standard for Software Maintenance*, 1998.

[11] ISO/IEC, *ISO/IEC 14764 : 2006, Software Engineering-Software Life Cycle Processes-Maintenance* (2nd ed.), 2006-09-01.

[12] ISO/IEC 25010 : 2011, *Software Engineering : Software Product Quality Requirements and Evaluation (SQuaRE) Quality Model and Guide*, ISO, 2011.

[13] ISO/IEC 25022.2 : 2015 *Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE) -Measurement of Quality in Use*, ISO, 2015.

[14] ISO/IEC 25030 : 2007, *Software engineering-Software product Quality Requirements and Evaluation (SQuaRE)-Quality Requirements*, ISO, 2005.

[15] Kemerer, C. F. and Slaughter, S. A., "Determinants of Software Maintenance Profiles : An Empirical Investigation", *Journal of Software Maintenance : Research and Practice*, Vol. 9, No. 4, 1997, pp. 235-251.

[16] Kitchenham, B. and Pfleeger, S. L., "Software Quality : The Elusive Target [special issue section]", *Software*, IEEE, Vol. 13, 1996, pp. 12-21.

[17] Koh, S. and Han, M. P., "Purposes, Results, and Types of Software Post Life Cycle Changes", *Journal of Information Technology Applications and Management*, Vol. 22, No. 3, 2015, pp. 143-167.

[18] Koh, S. and Whang, J., "A Critical Review on ISO/IEC 25000 SQuaRE Model", *Proceedings of the 15th International Conference on IT Applications and Management : Mobility, Culture and Tourism in the* Digitalized *World*, (ITAM15), 2016, pp. 42-52.

[19] Lientz, B. and Swanson, B., *Software Maintenance Management*, Addison-Wesley, Reading, MA, 1980.

[20] Sneed, H. M., "A Cost Model for Software Maintenance & Evolution", *Proceedings of the 20th IEEE International Conference on Software Maintenance* (ICSM'04), 2004.

<Appendix>

Definitions[1] of ISO/IEC 25010 : 2011 Product Quality Characteristics and Sub-Characteristics

- Compatibility : degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software **environment**
  - Co-existence : degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product
  - Interoperability : degree to which two or more systems, products or components can exchange information and use the information that has been exchanged.
- Functional suitability : degree to which a product or system provides functions that meet stated and implied needs when used under specified **conditions**.
  - Functional appropriateness : degree to which the functions facilitate the accomplishment of specified tasks and objectives.
  - Functional completeness : degree to which the set of functions covers all the specified tasks and user objectives.
  - Functional correctness : degree to which a product or system provides the correct results with the needed degree of precision.
- **Maintainability** : degree of effectiveness and efficiency with which a product or system can be **modified** by the intended maintainers
  - **Analyzability** : degree of effectiveness and efficiency with which it is possible to **assess** the impact on a product or system of an intended **change** to one or more of its parts, or to **diagnose** a product for deficiencies or causes of failures, or to **identify** parts to be modified
  - **Modifiability** : degree to which a product or system can be effectively and efficiently **modified** without introducing defects or degrading existing product quality
  - Modularity: degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components
  - **Reusability** : degree to which an asset can be <u>used</u> in more than one system, or in **building** other assets
  - **Testability** : degree of effectiveness and efficiency with which test criteria can be **established** for a system, product or component and **tests** can be performed to determine whether those criteria have been met

1) All the definitions and notes in this appendix are cited unchanged from their original expressions.

- Performance efficiency : performance relative to the amount of resources used under stated **conditions**.
  - Capacity : degree to which the maximum limits of a product or system parameter meet requirements.
  - Resource utilization : degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.
  - Time behavior : degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.
  - information and use the information that has been exchanged
- **Portability** : degree of effectiveness and efficiency with which a system, product or component can be **transferred** from one hardware, software or other operational or usage **environment** to another
  - **Adaptability** : degree to which a product or system can effectively and efficiently be **adapted** for different or evolving hardware, software or other operational or usage environments
  - **Installability** : degree of effectiveness and efficiency with which a product or system can be successfully **installed and/or uninstalled** in a specified **environment**
  - Replaceability : degree to which a product can replace another specified software product for the same purpose in the same **environment**.
- Reliability : degree to which a system, product, or component performs specified functions under specified **conditions** for a specified period of time.
  - **Availability** : degree to which a system, product, or component is operational and **accessible** when required for use.
  - Fault tolerance : degree to which a system, product, or component operates as intended despite the presence of hardware or software faults.
  - Maturity : degree to which a system meets need for reliability under normal operation.
  - Recoverability : degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.
- Security : degree to which a product of system protects information and data so that persons or other products of systems have the degree of data access appropriate to their types and levels of authorization.
  - Accountability : degree to which the actions of an entity can be traced uniquely to the entity.
  - Authenticity : degree to which the identity of a subject or resource can be proved to be the one claimed.
  - Confidentiality : degree to which a product of system ensured that data are accessible only to those authorized to have access.
  - Integrity : degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data.
  - Non-repudiation : degree to which actions or events can be proven to have taken places, so that the events or actions cannot be repudiated later.

- **Usability** : degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified **context** of use.
  - **Accessibility** : degree to which a product or system can be **used** by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified **context** of use.
  - Appropriateness **recognizability** : degree to which users can **recognize** whether a product or system is appropriate for their needs.
  - **Learnability** : degree to which a product or system can be used by specified users to achieve specified goals of **learning to use** the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified **context** of use.
  - **Operability** : degree to which a product or system has attributes that make it easy to **operate and control**
  - User error protection : degree to which a system protects users against making errors.
  - User interface aesthetics : degree to which a user interface enables pleasing and satisfying interaction for the user

Author Profile

Seokha Koh

Seokha Koh is the professor of the Department of MIS, Chungbuk National University. His current primary research areas include Software Quality Management, Business Process Modeling, Software Architecture, Project Management, and Software Engineering.