J. Inf. Technol. Appl. Manag. 26(2): 111~123, April 2019
https://doi.org/10.21219/jitam.2019.26.2.111

ISSN 1598-6284 (Print)
ISSN 2508-1209 (Online)

An Activity-Centric Quality Model of Software

Seokha Koh*

Abstract

In this paper, software activity, software activity instance, and the quality of the activity instance are defined as the 'activity which is performed on the software product by a person or a group of persons,' the 'distinctive and individual performance of software activity,' and the 'performer's evaluation on how good or bad his/her own activity instance is,' respectively. The representative values of the instance quality population associated with a product and its sub-population are defined as the (software) activity quality and activity quality characteristic of the product, respectively. The activity quality model in this paper classifies activity quality characteristics according to the classification hierarchy of software activity by the goal. In the model, a quality characteristic can have two types of sub-characteristics : Special sub-characteristic and component sub-characteristic, where the former is its super-characteristic too simultaneously and the latter is not its super-characteristic but a part of its super-characteristic. The activity quality model is parsimonious, coherent, and easy to understand and use. The activity quality model can serve as a corner stone on which a software quality body of knowledge, which constituted with a set of models parsimonious, coherent, and easy to understand and use and the theories explaining the cause-and-relationships among the models, can be built. The body of knowledge can be called the (grand) activity-centric quality model of software.

Keywords : Software Quality, Software Activity, Software Activity Quality, Software Quality View, Cause And Effect Relationships

Received : 2019. 02. 28. Final Acceptance : 2019. 05. 09.

* Department of Management Information Systems, Chungbuk National University, 1 Chungdae-ro Seowon-gu Cheongju, Chungbuk, 28644, Korea, Tel : +82-43-261-7062, e-mail : shkoh@cbnu.ac.kr

1. Introduction

This paper presents an activity quality model which is a direct extension of Koh and Koh's [2018] activity-oriented usability model for software. Koh and his colleagues [Koh 2016, 2017a, 2017b; Koh and Jiang, 2017] define software activity as the activity which is performed on the software product by a person or a group of persons. Using, changing, and installing software products, for example, are important types of software activity. A software product should be good for various software activity types.

In this paper, the activity type and the activity instance are distinguished. Using, changing, developing, and installing, for example, will be used to denote types of software activity. Using instance, changing instance, developing instance, and installing instance, for example, will be used to denote the distinctive and individual performance of corresponding type of software activity. Rigorously expressing, whenever a person or a group of persons performs an activity on a software product to achieve distinctive and specific objectives, an activity instance is generated. For example, a specific individual's playing a specific software game product or shopping on a specific internet shopping mall at some specific moment is a using instance.

The quality of something is can be defined as, according to Collins Cobuild Advanced Learner's English Dictionary, '*how good or bad it is*.'[1] Koh and Koh [2018] define the quality of a software product as 'how product good or bad is the product?' In this paper, the activity quality of a software product is defined as "how good or bad is the product for software activity?'

The goodness of a software product may vary by software activity type. Specifically, Koh and Koh [2018] define the usability of a software product as 'how good a product is for being used' or 'goodness for using.' This definition of usability is very simple and precise in sharp contrast with many existing definitions of other authors, for example, ISO/IEC SQuaRE (ISO/IEC 25000 Series, Systems and Software Quality Requirements and Evaluation).

Software quality is a very confusing concept. For example, some authors regard usability concerning user's experiences [Gonzalez Sanchez et al., 2009b; Microsoft Corporation 2000; U.S. Department of Health and Humanity Services, 2017] while other authors regard it concerning the product itself [Herrera et al., 2010; ISO 9126-1, 9241-11, 25000 Series; Nielsen, 2016]. Koh and his colleagues [Koh, 2016, 2017a, 2017b; Koh and Jiang, 2017; Koh and Koh, 2018] combine these two approaches to define two types of usability : the usability instance as the goodness evaluated by the user him/herself of an individual using instance, and the product usability as the goodness of the product for using as a type of software activity. The product usability is get by aggregating all usability instances associated with a product.

In section 2, a classification model of software activity types is presented. In section 3, an activity software quality model is presented. The parts regarding using and usability in sections 2 and 3 correspond to those in Koh and Koh's [2018] activity-oriented usability model of software. In section 4, the discussions for a more comprehensive software quality body of knowledge are presented. In section 5, the conclusions and suggestions for further research are presented.

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

2. Software Activity Model

In this paper, the sub-types of a software activity type (the super-type activity) are classified into two categories :

- **Special Sub-type** : An activity instance of a special sub-type is an activity instance of the super type too.
- **Component Sub-type** : An activity instance of a component sub-type is not a super-type activity instance, but a part of a super-type activity instance.

〈Figure 1〉 shows an example of the classification of special sub-types of software activity. The criterion of the classification is the goal of the activity. The classification is not mutually exclusive. That is, it is possible to classify an activity instance to more than two types simultaneously. However, the level 1 classification is designed to be exhaustive. That is, it is possible to classify every activity instance into at least one type of level 1 properly.

It is noticeable that accessing is separated from using since accessing can occur independently with using : For example, A person may access a software product, for example, to test or study it, or customize its interface [Koh and Jiang, 2017].

It is also noticeable that learning, recognizing appropriateness and testing are absent in 〈Figure 1〉. Testing is classified as a component sub-type of changing (refer 〈Figure 2〉).

Learning and recognizing is not regarded as a type of software activity since they are generally used to denote mental phenomena which occur inside a person [Koh, 2017a]. For example, Oxford Learner's Dictionary defines learning as '*gaining knowledge or skill by studying, from experience, from being taught*. On the other hand, studying or examining are

Developing : (Developer)* Making a usable software product.

Operating : (Operator) Letting a software product able to be used.
- √ Installing : Making the product operative initially.
- √ Maintaining : Keeping the product in operation.
- √ Recovering : Making the product operative again

Examining : (Potential or Entry-level User) Studying to learn about the product or practicing using the product.

Accessing (User) : Becoming to be able to use the product.

Using (User) : Interfacing with the product with specific goals other than examining and changing the product.**
- √ Working : Using a product to complete a set of predetermined tasks.
- √ Studying : Using a product to learn about a particular subject or subjects.
- √ Play : using a product to enjoy.

Changing (User or Software Engineer) : Making the product altered.
- √ Changing user-interface : Altering user-interface
- √ Changing data-interface : Altering data-interface.
- √ Changing data processing : Altering the processing of data.
- √ Changing architecture : Altering architecture.

Retiring (Operator) : Letting the product unable to be used any longer.

* (the type of stakeholder who performs the corresponding activity type).
** This part is the same as the corresponding part of Koh and Koh [2018].

〈Figure 1〉 Level 1 and Level 2 Special Sub-Types of Software Activity

usually used to denote a activity type that take time and efforts. Examining and studying are distinguished by their goals : learning about the product itself or some other things.

The classification in 〈Figure 1〉 is not mutually exclusive. For example, a person can interface with a software product both for using and examining the product.

Maintaining is one of the most confusing software terminologies. For a hardware product, maintaining is typically used to mean preserving and restoring its original state [Koh and Han, 2015; NF EN 13306 : 2001]. For a software product, however, maintaining is typically used to refer changing or modifying it [ANSI/IEEE Std. 729 : 1983; Boehm, 1981; IEEE Std. 1219 : 1998; ISO/IEC 14764 : 2006; Martin and Osborne, 1983]. GAO (The American General Accounting Office) and ANSI/IEEE Std. 729 : 1983 define software maintenance to include virtually all changes made on a software product after its delivery. Although the definitions of these organizations are generally accepted as the standard definitions of software maintenance, some literatures define software maintenance as a special type of software modification or change [Abran and Nguyenkim, 1993; Chaptin et al., 2001; Hatton, 2007; Hunt et al., 2008; ISO/IEC 14764 : 2006; Koh and Han, 2015; Martin and Osborne, 1983; Sneed, 2004]. It is one of the most important causes of the chaos in the literature and practice regarding software maintenance to use the term 'software maintenance' without mentioning its scope explicitly [Chaptin et al., 2001; Koh and Han, 2015; Seed, 2004]. Here, maintaining is used to mean preserving the initial installed state of the product and do not involve changing the source code of the product.

Changing in this paper denotes changing a software product after its delivery. Changing during the development phase is not included in the 'changing.' Changing user-interface, data-interface, data processing, and architecture are special types of changing. Special sub-types of changing may be classified otherwise. For example, Koh and Han [2015] classify the post life cycle changes of software classify into modification, enlargement, replacement, and retirement. It is beyond the scope of this paper, however, to classify the special sub-types of changing rigorously. The goal of this section is to present a satisfactory practical example of the classification of software activity.

Using (User) : Interfacing with the product with specific goals other than examining and changing the product.**
- √ Navigating : Moving from a page to another
- √ Data-Preparing : Preparing the data to be input
- √ Data-Inputting : Inputting the data prepared.
- √ Response-Waiting : Waiting for the response of the product.
- √ Output-Examining : Examining the output of the product.
- √ Output-Utilizing : Utilizing the output examined.

Changing (User or Software Engineer) : Making the product altered.
- √ Analyzing : Preparing to alter the product (validly, effectively and efficiently).
- √ Changing source code : Altering the source code of the product.
- √ Testing : Identifying the faults in the altered product.

* (the type of stakeholder who performs the corresponding activity type).
** This part is the same as the corresponding part of Koh and Koh [2018].

〈Figure 2〉 Examples of Component Sub-Type of Using and Changing

〈Figure 2〉 shows examples of component sub-type of using and changing. Especially, the component sub-types of using are intended to be exhaustive and mutually exclusive. That is, every atomic activity of a using instance can and should be classify into one and only one of them. Those of changing are not intended to be so rigorous. Especially, testing is generally not performed against the current product, but generally performed against the altered product. So, it may improper to define testing as the software activity that is performed to the (current) software product.

3. Goodness of a Software Product

In this paper, the representative values of the instance quality population associated with a product and its sub-population are defined as the (software) activity quality and activity quality characteristic of the product, respectively. Koh and Koh's [2018] usability corresponds to the population of usability instances associated with a software product.

According to Oxford online dictionary, the suffix '-ability' means 'the quality of being able to be or having to be' or 'the fact of having the quality mentioned.' According to this principle, for example, usability can be used to mean 'the quality of being able to be used,' 'how good it is to be used,' or 'goodness for using.' [Koh and Whang, 2016; Koh and Jiang, 2017; Koh and Koh, 2018]. Applying this naming scheme, the activity quality characteristics associated with software activity types on 〈Figure 1〉 can be defined as in 〈Figure 3〉 Koh and Whang's [2016] '-ability' principle can be rephrased as the following.

Developability : (Developer; Requirement documents and/or specifications)* Goodness for developing.

Operate-ability : (Operator; Source code or Executable code) Goodness for operating.
- √ Install-ability : Goodness for installing.
- √ Maintainability : Goodness for maintaining.
- √ Recoverability : Goodness for recovering

Examinability : (Potential or Entry-level User, Manager, Software Engineer; Source code and other documents) Goodness for examining.

Access-ability (Potential user; Executable code with user interfaces) : Goodness for accessing.

Usability (User; Binary code with user interfaces) : Goodness for using.**
- √ Work-ability : Goodness for working,
- √ Study-ability : Goodness for studying.
- √ Play-ability : Goodness for playing,

Changeability (User or Software Engineer, Manager; Source code) : Goodness for changing.
- √ User-interface change-ability : Goodness for changing user-interface.
- √ Data-interface change-ability : Goodness for changing data-interface.
- √ Data-processing change-ability : Goodness for changing data processing.
- √ Architecture change-ability : Goodness for changing architecture.

Retire-ability (Operator; Executable code and associated things) : Goodness for retiring.

* (The type of stakeholder who performs and evaluate the corresponding activity instance; Object of evaluation).
** This part is the same as the corresponding part of Koh and Koh [2018].

〈Figure 3〉 Level 1 and Level 2 Special Characteristics of Software-Activity Quality Model

Usability*
- √ Navigate-ability : Goodness for navigating,
- √ Data-prepare-ability : Goodness for preparing data to be input,
- √ Data-input-ability : Goodness for inputting data,
- √ Response-wait-ability : Goodness for waiting response,
- √ Output-examine-ability : Goodness for examining output data.
- √ Output-utilize-ability : Goodness for utilizing the output. data

Change-ability.
- √ Analyzability : Goodness for analyzing.
- √ Source-code change-ability : Goodness for changing the source code.
- √ Testability : Goodness for testing.

*This part is the same as the corresponding part of Koh and Koh [2018].

〈Figure 4〉 Component Sub-Characteristics of Usability and Change-Ability

- **Activity quality naming principle** : The suffix '-ability' can be attached to and only to software activity type to denote the corresponding activity quality characteristics.

As activity types, an activity quality characteristic can have two types of sub-characteristics :
- **Special sub-characteristic** : The sub-characteristic of this type is its super-characteristic simultaneously.
- **Component sub-characteristic** : The sub-characteristic of this type is not its super-characteristic simultaneously. The aggregation of sub-characteristic of this type can be used as the super characteristic.

〈Figure 4〉 shows the component sub-characteristics of usability and change-ability corresponding to their component sub-types. It will provide much richer and more reliable data to measure the component sub-characteristics and to aggregate them than to measure the usability or change-ability directly [Koh and Koh, 2018].

Significant special sub-characteristics of an activity quality characteristic can be measured and aggregated to be the activity quality characteristic. It also will provide much richer and more reliable information than to measure the activity quality characteristic directly. For example, study-ability and play-ability can be measured respectively and aggregated to usability for an educational game product [Koh and Koh, 2018]. If play-ability is turned out to be low relatively to study-ability, then effort can be devoted chiefly into enhancing entertainingness.

4. Discussions

Koh and his colleagues [Koh, 2017a, 2017b; Koh and Jiang, 2017; Koh and Koh, 2018] have proposed a model of the view (refer 〈Figure 5〉) and quality view and principles regarding software quality :
- **Principle of one view** : A software quality model should correspond to one and only one software quality view.
- **Principle of Parsimony** : The quality characteristic should be defined as brief and precise as possible.
- **Principle of Cohesiveness** : The quality characteristic should consist of elements that fit together well and form a united whole.
- **Principle of Inheritance** : Every aspect of usability should be able to be inherited by its special sub-characteristics, possibly, with proper specializations.

- **End view** : It represents the effort to find out for what the software product should be good. The quality characteristics in this view correspond to the effects of good quality in means view.
  - ▸ **Short-term view** : It focuses on short-term effects of various software activity types.
    - √ **Performer's view on software activity** : It represents the performer's subjective evaluation of the software activity that he/she has performed.
    - √ **Third party's view on software activity** : It represents the interests of stakeholders other than the performer, which are associated with individual software activities.
  - ▸ **Long-term view** : It focuses on the long-term and aggregated effects on various stakeholders.
- **Means view** : It represents the effort to make the software product good for various ends. The elements of this view correspond to the causes of desirable effects. Software engineers should be able to manipulate the elements to improve the quality in end view.
  - ▸ **Intrinsic view** : It identifies static and invariant properties of the software product, which affect the achievement of ends. It does not change unless the product is changed.
  - ▸ **Contingency view** : It identifies static and invariant emerging properties of contingencies, which affect the achievement of ends. It can change even if the product is not changed.

〈Figure 5〉 Koh and Jiang's [2017] Model of Software Quality View

The activity software quality model in section 3 abides by all the four principles. Specifically, it corresponds to the 'performer's view on software activity' in 〈Figure 5〉. As the result, the activity quality model is parsimonious, coherent, and easy to understand and use.

Most of existing software quality model violate the principle of one view. For example, SQuaRE suffers from ambiguity, inconsistency, and contradictions in the definitions of quality characteristics and sub-characteristics, making it un-suitable to measure the design quality of software product [Al-Kilidar, 2005; Haboush et al., 2014; Kitchenham and Pfleeger, 1996; Koh, 2017a; 2017b; Koh and Jiang, 2017; Koh and Whang, 2016].

Koh [2017] reclassifies 86 measures which belong to 8 characteristics (31 sub-characteristics) into 7 categories (15 sub-categories) : Data processing (valid; correct; efficient), data interface, user interface (information providing, data inputting; process controlling; presentation, appearance & composite), composite, software engineering (diagnosis and testing; architecture; coding), system (capacity; runtime behavior; operation management), activity view (developing & maintaining; installing) (refer 〈Table 1〉). The result shows the fundamental reason of SQuaRE's problems : The items related with end and means exist mixed unsystematically in a model (refer 〈Table 1〉 and 〈Table 2〉). For example, usability of a product is influenced by data processing of the product and the system on which the product is operated as well as by user interface of the product. Usability is indirectly influenced by data interface of the product too. In SQuaRE, only the items related to user interface are classified into usability, implying implicitly that usability is influence only by user-interface.

Moreover, among 86 measures of the product quality model, only 21 measures are intrinsic [Koh, 2017b]. This means that the product quality model does not provide sufficient information for software engineers to improve the activity quality. It is the intrinsic quality attributes that software engineers can control to improve the activity quality. The contingency quality attributes are influenced many factors other than the product, which software engineers cannot control. More intrinsic quality attributes should be identified to complete a separate intrinsic

〈Table 1〉 Koh's [2017a] Classification of the Measures of SQuaRE's Product Quality Model

Category	Measures of SQuaRE's Product Quality Model
DP (data processing)-Valid	FAp* (functional appropriateness of usage objective, functional appropriateness of system), FCp (functional coverage)
DP-Correct	FCr (functional correctness)
DP-Efficient	ERu (bandwidth utilization, mean processor utilization, mean memory utilization, mean I/O devices utilization,)
Data interface	CIn (data formats exchangeability, data exchange protocol sufficiency, wxternal interface adequacy), PRe (data reusability/import capability)
UI (user interface)-Information providing	UAp (demonstration coverage, description completeness, entry point self-descriptiveness), ULe (error messages understandability, self-explanatory user interface, user guidance completeness), UOp (message clarity, monitoring capability)
UI-Data inputting	SIn (buffer overflow prevention), UEp (avoidance of user operation error, user entry error correction, user error recoverability), ULe (entry fields defaults), UOp (input device support)
UI-Process controlling	UOp (functional customizability, undo capability)
UI-Presentation, appearance & composite	UAc (accessibility for users with disabilities, supported languages adequacy), UIn (appearance aesthetics of user interfaces), UOp (appearance consistency, operational consistency, understandable categorization of information, user interface customizability)
Composite	CCo (co-existence with other products), PAd (hardware environmental adaptability, operational environment adaptability, system software environmental adaptability), PRe (functional inclusiveness, product quality equivalence, usage similarity)
SWE (SW engineering)-Diagnosis and testing	MAn (diagnosis function effectiveness, diagnosis function sufficiency), MTe (autonomous testability, test function completeness, test restartability)
SWE-Architecture	MMo (coupling of components, cyclomatic complexity adequacy), MRe (reusability of assets)
SWE-Coding	MRe (coding rules conformity)
System-Capacity	ECa (transaction processing capacity, user access capacity, user access increase adequacy), RFt (redundancy of components), ETb (mean response time, mean throughput, mean turnaround time, response time adequacy)
System-Runtime Behavior	MAn (system log completeness), RAv (mean downtime, system availability), RFt (mean fault notification time), RMa (failure rate, mean time between failure), ETb (turnaround time adequacy), RRe (mean recovery time)
System-Operation Management	PIn (ease of installation), RRe (backup data completeness), SAc (authentication mechanism sufficiency, authentication rules conformity, system log retention, user audit trail completeness), SCo (access controllability, data encryption correctness, strength of cryptographic algorithms), SIn (data integrity, internal data corruption prevention), SNo (digital signature usage)
AV (activity view)-Developing & Maintaining	MMd (modification capability, modification correctness, modification efficiency), RFt (failure avoidance, fault correction), RMa (test coverage)
AV-Installing	PIn (installation time efficiency)

*The sub-characteristic to which the measures in the parenthesis belong. For the abbreviation, refer <Table 2>.

quality model of software. The model should conform to the principles suggested above.

The quality model subjects to contingency view, third party's view, and long-term view respectively should be developed too. They should confirm to the principles suggested above. The theories to explain the cause-and-effect relationships among the models should be developed too. The theories and the models will constitute a software quality body of knowledge which will guide development, use, and post life cycle changes of software.

〈Table 2〉 Correspondence between the Activity Quality Model and the Product Quality Model of SQuaRE

This Paper : Level 1 Characteristic	Product Quality Model of SQuaRE			
	Characteristic		Measure	
	Short-term View	Mean View	Short-term View	Mean View
Developability			Activity view[i]	
Operate-ability	P, PIn, PRe; S, SAc, SAc, SNo; UOp	SCo, SIn	Activity view[ii] System[iii]	Composite[iv] Data interface, System[v]
Examinability				Composite[vi] User interface[vii]
Access-ability				System[viii]
Usability	U, UAc	C, CCo, CIn; F, FAp, FCp, FCr; E, ECa, ERu, ETb; UIn,		Data processing; System[ix] User interface[x]
Changeability	M, MAn, MMd, MRe, MTe; PAd	MMo	Activity view[i]	SW Engineering
Retire-ability				
None	UAp, ULe			Composite[xi]

* Abbreviation of characteristics (sub-characteristics) : For example, CCo means 'Co-existence of Compatibility.
C (Co, In) = Compatibility (Co-existence, Interoperability),
E (Ca, Ru, Tb) = Performance Efficiency (Capacity, Resource utilization, Time behavior)
F (Ap, Cp, Cr) = Functional suitability (Functional appropriateness, Functional completeness, Functional correctness),
M (An, Md, Mo, Re, Te) = Maintainability (Analyzability, Modifiability, Modularity, Reusability, Testability),
P (Ad, In, Re) = Portability (Adaptability, Install-ability, Replaceability),
R (Av, Ft, Ma, Re) = Reliability (Availability, Fault tolerance, Maturity, Recoverability)
S (Ac, Au, Co, In, No) = Security (Accountability, Authenticity, Confidentiality, Integrity, Non-repudiation),
U (Ac, Ap, Ep, In, Le, Op) = Usability (Accessibility, Appropriateness recognizability, User error protection, User interface aesthetics, Learnability, Operability)

i. All the items belong to Developing & maintaining in <Table 5>. It is noticeable these items are related to change-ability too.
ii. Installing
iii. Operation management (including ease of installation only; It belongs to PIn in SQuaRE)
iv. Excluding product quality equivalence, usage similarity.
v. Operation management (excluding ease of installation).
vi. Including usage similarity only; It belongs to PRe in SQuaRE.
vii. Information providing (excluding monitoring capability), presentation, appearance, & composite (including appearance consistency only; It belongs to UOp in SQuaRE)
viii. Capacity (including redundancy of component, user access capacity, user access increase adequacy only; They belong to RFt, ECa, ECa in SQuaRE, respectively), runtime behavior (excluding system log completeness, mean fault notification time).
ix. Capacity (Excluding redundancy of component, user access capacity, user access increase adequacy), time behavior (including system log completeness, mean fault notification time only; They belong to MAn, RFt in SQuaRE, respectively).
x. Excluding the measures that is related to examinability.
xi. Including product quality equivalence only which belongs to PRe in SQuaRE.

5. Conclusions

The activity quality model in this paper is a direct extension of Koh and Koh's [2018] activity-oriented usability model for software. It has 7 characteristics, developability, operate-ability, examinability, access-ability, usability, changeability, and retire-ability, which correspond to developing, operating, examining, accessing, using, changing, and retiring, respec-

tively. They are designed to be exhaustive and mutually exclusive.

A quality characteristic can have two types of sub-characteristics : Special sub-characteristic and component sub-characteristic, where the former is its super-characteristic too simultaneously and the latter is not its super-characteristic but a part of its super-characteristic. For example, usability has work-ability, study-ability, and play-ability as its special sub-characteristics. This classification of special sub-characteristics of usability is really an example and is meant to be neither exhaustive nor mutually exclusive. Usability has navigate-ability, data-prepare-ability, data-input-ability, response-wait-ability, output-examine-ability, and output-utilize-ability as its component sub-characteristics. They are meant to be exhaustive and mutually exclusive. The activity quality (sub-)characteristic is defined as "how good or bad is the product for the corresponding software activity?' or, shortly, 'goodness for the corresponding software activity.'

The activity quality model is parsimonious, coherent, and easy to understand and use. The activity quality model can serve as a corner stone on which a software quality body of knowledge, which constituted with a set of models parsimonious, coherent, and easy to understand and use and the theories explaining the cause-and-relationships among the models, can be built. The body of knowledge can be called the (grand) activity-centric quality model of software. To be so, the quality models which are subjects to contingency view, third party's view, and long-term view respectively should be developed. They should confirm to the principles of one view, parsimony, cohesiveness, and inheritance. The theories to explain the cause-and-effect relationships among the models should be developed thereafter.

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

[3] Boehm, B. W., *Software Engineering Economics*, Prentice Hall PTR : Upper Saddle River, NJ, USA, 1981.

[4] Chaptin, N., Hale, J., Kahn, K. R., Tan Jr. W. G., "Types of Software Evolution and Software Maintenance", *Journal of Software Maintenance and Evolution*, Vol. 13, No. 1, 2001, p. 3.

[5] Collins Cobuild Advanced Learner's English Dictionary, reference date : 05/16/2016.

[6] González Sánchez, J. L., Padilla Zea, N., and Guitierrez Vela, F. L., "Playability : How to Identify the Player Experience in a Video Game", Proceedings of IFIP Conference on Human-Computer Interaction : Human-Computer Interaction-INTERACT 2009, pp. 356-359.

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

[8] Herrera, M., Moraga, M. A., Caballero, I., and Calero, C., "Quality in Use Model for Web Portals (QiUWeP)", In : Daniel F., Facca F. M. (eds), *Current Trends in Web Engineering. ICWE 2010, Lecture Notes in Computer Science*, Vol. 6385, Springer : Berlin, Heidelberg, 2010, pp. 91-101.

〔9〕 Hunt, B., Turner, B., and McRitchie, K., "Software maintenance Implications on Cost and Schedule", *Proceedings of Aerospace Conference*, 2008 IEEE, 2008, pp. 1-8.

〔10〕 IEEE, *IEEE Std. 1219-1998, IEEE Standard for Software Maintenance*, 1998.

〔11〕 ISO/IEC, *ISO/IEC 14764 (IEEE Std 14764-2006), Software Engineering-Software Life Cycle Processes-Maintenance* (2nd ed.), 2006-09-01.

〔12〕 ISO/IEC 25010 : 2011, *Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models*, ISO, 2011.

〔13〕 ISO/IEC 25023 : 2016, *Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-Measurement of system and Software Product Quality*, ISO, 2016.

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

〔15〕 Koh, S., "Cause-and-Effect Perspective on Software Quality : Application to ISO/IEC 25000 Series SQuaRE's Product Quality Model", *Journal of Information Technology Applications and Management*, Vol. 23, No. 3, 2016, pp. 71-86.

〔16〕 Koh, S., "The Checklist for System and Software Product Quality Implied in the Product Quality Model of ISO/IEC 25000 Series SQuaRE", Proceedings of 17th International Conference on IT Applications and Management : Babolsar, Iran, 22-23 February 2017a, pp.126-136.

〔17〕 Koh, S., "The Principle of One Quality View and Division of Product Quality Model of ISO/IEC 25000 Series SQuaRE", *Asian Journal of Information and Communications*, Vol. 9, No. 1, 2017b, pp. 87-101.

〔18〕 Koh, S. and Han, M. P., "Purposes, Results, and Types of Software Post Life Cycle Changes", *Journal of IT Applications and Management*, Vol. 22, No. 3, 2015, pp. 143-167.

〔19〕 Koh, S. and Jiang, J., "What should Using a Software Product and Usability of the Software product be?", *Journal of Information Technology Applications and Management*, Vol. 24, No. 3, 2017, pp. 73-92.

〔20〕 Koh, S., Youjoung Koh, "The Activity-Oriented Usability Model od Software", *Journal of IT Applications and Management*, Vol. 25, No. 3, 2018, pp. 17-28.

〔21〕 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.

〔22〕 Martin, R. J. and Osborne, W., "Guidance of Software Maintenance", *U.S. National Bureau of Standards*, NBS Pub., 1983, pp. 500-129,

〔23〕 Microsoft Corporation, "Usability in Software Design", https://msdn.microsoft.com/en-us/library/ms997577.aspx, 2000 (reference date : 17/04/2017).

〔24〕 NF EN 13306, *Terminologies de la Maintenance*, June 2001.

〔25〕 Nielsen, J., "Usability 101 : Introduction to Usability", Nielsen Norman Group, https://www.nngroup.com/articles/usability-101-introduction-to-usability/, Jan 4, 2012 (reference data 17/04/2017).

〔26〕 Oxford Learner's Dictionary, reference date : 05/16/2016.

〔27〕 Sneed, H. M., "A Cost Model for Software Maintenance and Evolution", *Proceedings*

of the 20th IEEE International Conference on Software Maintenance (ICSM'04), 2004, pp.264-273.

[28] U.S. Department of Health & Humanity Services, "Usability Evaluation Basics", https://www.usability.gov/what-and-why/usability-evaluation.html, reference date : 17.04/2017.

저자소개

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.