Journal of Theoretical and Applied Information Technology
30th June 2018. Vol.96. No 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A MEASUREMENT MODEL OF THE FUNCTIONAL SIZE OF SOFTWARE MAINTAINABILITY REQUIREMENTS

[1]KHALED ALMAKADMEH, [2]KHALID T. AL-SARAYREH, [3]KENZA MERIDJI

[1]Assistant Professor, Department of Software Engineering, The Hashemite University, Jordan

[2]Associate Professor, Department of Software Engineering, The Hashemite University, Jordan

[3]Associate Professor, Department of Software Engineering, University of Petra, Jordan

E-mail: [1]khaled.almakadmeh@hu.edu.jo, [2]khalidt@hu.edu.jo, [3]kmeridji@uop.edu.jo

ABSTRACT

The European ECSS-E-40 standard for the aerospace industry includes maintainability as one of sixteen non-functional requirements for the embedded and real time software. The software maintainability requirements measured internally and externally. According to the ECSS European standards, maintainability requirements are apportioned to set maintainability requirements for lower level products to conform to the maintenance concept and maintainability requirements of the system and the maintainability analysis shall identify the maintainability critical items. This paper propose a new measurement model of the functional size of maintainability requirements of software. This functional size of the maintainability requirements measured using the concepts of the ISO19761: COSMIC standard at an early phase of the software development life cycle. It used as one of the primary inputs for the software effort estimation process. Further, this paper presents the design of software standard etalon to help in development of software products more effectively. An experiment is conducted to verify the applicability of the proposed measurement model to measure the functional size of requirements specifications of an online library software.

Keywords: *Maintainability Requirements, ISO19761, ECSS standards, Measurement method*

1. INTRODUCTION

Software measures are used as mechanisms to quantify several aspects of software product, process and projects. Software measures are used for different purposes including the assessment of software quality [1, 2], estimation of complexity [3], estimation of cost and effort [4, 5] as well as controlling the improvement process [6]. In spite of the existence of large number of software measures, the majority of them are unsuccessful [7] due to a number of weaknesses. For instance, software measures are usually defined informally [8], incomplete and/or inaccurate [9]. Therefore, such measures do not produce the required information estimation purposes.

The ECSS European International Standards [10, 11] and [12] present software maintainability as a non-functional requirement for embedded software. The ECSS standard [10] is a cooperative effort of the European space agency, the national space agencies and European industry associations for the purpose of developing and maintaining common standards. The international standard ECSS-E-40 part-1 B [10] addresses the management, engineering and product assurance in space projects and applications. This part of the standard is a level two standard: it is derived from and ECSS-E-40 [12] and ISO12207 [13] for space projects and is concerned with producing software (i.e. software that is part of a space system product-tree and developed as part of a space project).

According to the ECSS-E-40 part-1 B [10], the maintenance process contains the activities and tasks of the maintainer. The objective is to modify an existing software product while preserving its integrity. This process includes the migration and retirement of the software product. The process ends with the retirement of the software product. The maintainer manages the maintenance process at the project level by following the management process which is instantiated for software. This process consists of the following activities:

- Process implementation
- Problem and modification analysis
- Modification implementation

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- Conducting maintenance reviews
- Software migration
- Software retirement

The ECSS-Q-ST-30 part C [14] divides software maintainability requirements to set of requirements for lower level products to conform to maintenance concept and maintainability requirements of the system, and therefore, maintainability analysis shall identify maintainability critical items. The ISO/IEC 24765 [15] defines maintainability as the ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment. The IEEE 14764 [16] defines maintainability as the capability of the software product to be modified. Further, the IEEE 982.1 standard [17] defines maintainability as the speed and ease with which a program is corrected or is changed. In addition, the ISO25010 [18] standard defines maintainability as capability of a software product to be modified: such modifications to a software product include corrections, improvements or adaptation of software to changes in its environment, and in requirements and functional specifications.

The basic concepts and definitions of functional size measurement are standardized by ISO in [19]. Functional size measurement has come a long way, detailed descriptions of various functional size measurement methods are published as standards such as COSMIC [20], NESMA [21]. Functional size measurement is used for many purposes: for example to help estimating the effort of a starting development project or measuring the actual productivity of a finished development project. Other reasons of functional size measurement usage are presented in [22]. The COSMIC standard [20] defines the principles, rules and a process for measuring the functional size of a piece of software. The functional size is a measure of the amount of functionality provided by the software.

The paper presents the design of a measurement model to identify the functional size of software maintainability based on international standards and using COSMIC standard as a standardized method to measure the functional size of software maintainability requirements independently from the development languages technology. Therefore, this standardized measurement will overcome the weaknesses in the measurement of maintainability requirements presented in the literature.

The main contribution of this paper is a new measurement model to identify and measure maintainability requirements based on ISO19761 and ISO25010 international standard. The proposed measurement model represent a kind of a reference model in the sense of an 'etalon' standard used for measurement of maintainability. The measurement scope in this paper is to identify separately all the functionality allocated to software maintainability requirement as a piece of embedded software application, whether it has yet to be built or it has already been delivered.

This paper organized as follows: section 2 present the literature review, then section 3 present overview of the ISO19761: COSMIC international standard (i.e. ISO 19761) for functional size measurement of software. Section 4 present the design of the measurement model of maintainability requirements based on ISO international standards, and section 5 presents a quality evaluation of software maintainability requirements. Section 6 presents design of a software standard etalon. Finally, section 7 presents conclusions and future work directions.

2. LITERATURE REVIEW

Several research studies are conducted in the literature to measure the degree maintainability. Port and Taber [23] have reported an industrial study to emphasize the importance of developing maintainable software applications and the importance of planning the effort required to maintain such applications, especially critical software applications.

Wu et al. [24] have conducted a review on effort estimation approaches for maintenance of open source software applications. Twenty-nine approaches were identified for maintenance effort estimation; all presented estimation approaches use source code measures to calculate the maintenance effort needed for software applications.

Lin and Yeh [25] have proposed a software tool to calculate the functional size for source code of a software application under maintenance using the measurement rules and concepts proposed by the international standard for software functional size measurement. – ISO19761: COSMIC. Although these size measures are calculated using an ISO international standard, they are calculated using source line of code at a late phase of the software development life cycle.

Torkhan et al. [26] have proposed a conceptual interoperability framework to evaluate the degree of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

dependency between functional components in a software application. The proposed framework is built based on model-driven approach that analyze components that makeup a software application, then decompose these software components into low-coupling components while maintaining their interoperability.

Al-Saiyd [27] has proposed a bottom-up code comprehension model in order to analyze challenges that might face a software engineer in code comprehension, and to improve the readability of a software application source-code. The proposed model partitions a software application into several functional blocks at different levels of granularity in order to analyze their inter-dependencies using data and control flow graphs.

Manev and Dimov [28] have proposed a software tool to improve the documentation of software architecture to produce more maintainable software applications. The proposed tool analyze embedded software systems developed using C programming language and produce UML models that better presents the architectural details of such embedded systems.

Yan et al. [29] have proposed an aggregation method that automatically assigns weights to low-level measures of software quality in order to calculate more accurate high-level software maintainability characteristics. They applied a topic modeling technique to calculate probabilistic weight from a software benchmark.

Alhilman et al. [30] have combined five maintenance methods to improve maintenance policy of printing machines by reducing the need for manual calculation of low-level maintenance measures such as overall equipment effectiveness and reliability centered maintenance.

Gupta [31] has proposed an approach aimed to improve software maintenance through conducting a predictive analysis to identify shortcomings exist in business processes that are executed by software applications.

Malhotra and Chug [32] have conducted an empirical study to evaluate the impact of refactoring on software maintainability. The study is conducted by applying bad-smell refactoring methods on two versions of five proprietary software products (i.e. original and refactored version). The results of this study recommends that even though refactoring is a tedious process; the use of refactoring methods help to improve software quality and software maintainability in particular.

Mellegard et al. [33] have conducted an empirical study aimed to assess the impact of using domain-specific modeling in maintenance of a legacy system. The results of the study presented a positive impact of using domain-specific modeling in terms of early and low defect detection despite of the lengthy process and in terms of decreased maintenance effort needed to maintain this legacy system.

Plösch et al. [34] have proposed an automated tool for measurement of software maintainability. The tool calculates eighteen (18) measures of maintainability and is experimented using five open-source java projects. The calculated measures are then compared with measures calculated by EMISQ (expert centered method for internal software quality) quality model that calculates one-hundred and sixty-five (165) measures including maintainability measures.

Szőke et al. [35] have conducted an industrial study to investigate the impact of automatic refactoring on software maintainability. The study is conducted on four (4) industrial projects from four different companies to analyze maintainability changes resulted from different refactoring tasks using an automated tool that applies ColumbusQM quality model [36]. The refactoring analysis showed that almost all refactoring tasks had a consistent and traceable positive impact three of the four industrial projects and therefore reached more maintainable state.

Counsell et al. [37] conducted an empirical study to investigate relationship between maintainability index (MI) and object-oriented class features such as coupling, defects and size using two object-oriented software projects. A significant correlation is reported between class coupling and number of software defects.

In summary, several international standards (such as ISO and IEEE standards), European standards (i.e. ECSS standards) and research literature emphasized a high importance of developing software with managed complexity in order to produce maintainable software products. However, such research literature has measured the degree of maintainability to use it for effort estimation purposes at late phase of the software development lifer cycle, in which most project resources are already allocated and distributed. Whereas, such effort estimation such take place at an early phase of the development life cycle. Therefore, it is a crucial issue to obtain any early indicator of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

software maintainability requirements in order to build more accurate effort estimation models.

3. THE INTERNATIONAL STANDARD FOR SOFTWARE FUNCTIONAL SIZE MEASUREMENT: ISO19761

The ISO 19761 international standard [20] proposes a general model of software functional requirements that explains the borderline among hardware and software. This standardized method measures functional size of a software product independently of the technology used to develop such a product based on the identified functional user requirements. The COSMIC measurement method propose generic model of software functional user requirements in order to clarify the boundary between hardware and software. Figure 1 presents COSMIC model that demonstrate the generic flow of data from a functional perspective. In this model, software is typically bounded by hardware and it used either by a human user or by an engineered device. The human user interacts with software using a variety of input/output devices. Furthermore, software is bounded by storage hardware such as RAM memory.

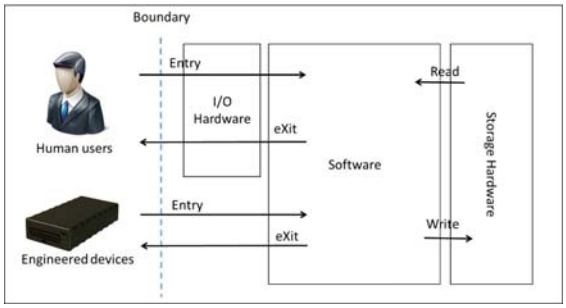

Figure 1 A generic model of ISO 19761 COSMIC measurement method

The functionality of software is enclosed within the data groups of functional flows. In order to specify these functional flows, four data movement types are identified by COSMIC as follows:

- Two data movement types (i.e. Entry and eXit) are identified specify the functional flows between human users and engineered devices from one side, and software from the other side.
- Two data movement types (i.e. Read and Write) are identified to specify the functional flows between storage and software.

Diverse perceptions are normally used for different measurement purposes. For example, in embedded and real time software, users are "engineered devices" interact straightforward with software. For business and management application software, the abstraction usually assumes that the users are one or more humans who interact directly with the business or management applications software across the border (the "I/O hardware" ignored). The ISO 19761 method is aimed to measure the size of software based on identifiable of functional user requirements. Then, they are allocated to hardware and software from the unifying perspective of a system integrating these two "components". Since the ISO 19761 standard is aimed at sizing software, only those requirements allocated to software are considered in its strategic measurement procedure.

4. DESIGN OF MEASUREMENT MODEL OF MAINTAINABILITY REQUIREMENTS

Four steps are recommended by Abran to carry out the design of a measurement model [38]:

4.1 Determination of Measurement Objectives

The objective: is to measure the functional size of the maintainability requirements as defined in ECSS-E Part 1B/2B and ECSS-Q-80B, ISO 25010 and using the ISO 19761 COSMIC standard as a measurement method. The measurement point of view is software perspective and the intended uses of the measurement results throughout the software life cycle: the functional size of the maintainability for a software product, whether it has yet built or it has already delivered.

4.2 Characterization of the Concept Measured

Definition of the concept to be measured: is the functional size of maintainability requirement; the maintainability measurements can be internal or external. Although the ECSS standard deals with maintainability specific to software-embedded system developed as part of a space project, the proposed measurement model is applicable for maintainability of non-embedded software product. The maintainability requirements are defined as the ease with which a software system or component can be modified to change or add capabilities, correct faults or defects, improve performance or other attributes, or adapt to a changed environment. The ISO 25010 [18] define the maintainability as the capability to modify a software product, these modifications include corrections, improvements or adaptation of software to changes in environment and in requirements and functional specifications. In ISO25010, there are two types of measures for maintainability requirements:

- External maintainability measures: should be able to measure such attributes as the behaviour

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of the maintainer, user, or system including the software, when the software maintained or modified during testing or maintenance.

- Analyzability should be able to measure such attributes as the maintainer or user effort or spent of resources when trying to diagnose deficiencies or causes of failures, or for identifying parts modified.
- Changeability should be able to measure such attributes as the maintainer or user effort by measuring the behaviour of the maintainer, user or system including the software when trying to implement a specified modification.
- Stability should be able to measure attributes related to unexpected behaviour of the system including the software when the software is tested or operated after modification.
- Testability should be able to measure such attributes as the maintainer or user effort by measuring the behaviour of the maintainer, user or system including software when trying to test the modified or non-modified software.

- Internal maintainability measures: used for predicting the level of effort required for modifying the software product.
  - Analyzability indicate a set of attributes for predicting the maintainer or user spent effort or spent resources in trying to diagnose for deficiencies or causes of failure, or for identification of parts to be modified in the software product.
  - Changeability indicate a set of attributes for predicting the maintainer or user spent effort when trying to implement a specified modification in the software product.
  - Stability indicates a set of attributes for predicting how stable the software product would be after any modification.
  - Testability indicates a set of attributes for predicting the amount of designed and implemented autonomous test aid functions present in the software product.

4.3 Identification of Maintainability Entity Types and Relationships among Entities

This part presents the identification of software maintainability entity types and the relationships among such entity types. Twelve entity types are identified to help software engineers to identify software maintainability requirements based of ISO international standards. Furthermore, this part presents four metamodels in order to capture the external and internal software maintainability requirements. A metamodel is an effective candidate to present visually different entity types, existing relationships, rules and constraints of a requirement-modeling problem.

4.3.1 Metamodel of Software Analyzability Requirements

There are four entity types to capture the analyzability requirements; audit trial capability, diagnostic function support, failure analysis capability, and status-monitoring capability. Figure 2 presents a metamodel that represents the four identified entity types and their corresponding relationships. This metamodel represent the relationship between entity types in terms of input, process and output.

Entity Type 1 (external measurement for analyzability)

- Entity name: audit trial capability
- Input of entity type 1: planned data recorded during operation
- Output of entity type 1: actual data recorded during operation
- Entity type 1 measures the functional size of audit trial capability
- Entity relationship: many-many recorded data on the system

Entity Type 2 (internal measurement for analyzability)

- Entity name: diagnostic function support
- Input of entity type 2: failure causes
- Output of entity type 2: actual registered failure
- Entity type 2 measures the functional size of diagnostic function support
- Entity relationship: many-many failures types in the system

Entity Type 3 (external measurement for analysability)

- Entity name: failure analysis capability
- Input of entity type 3: failure diagnoses
- Output of entity type 3: actual registered failure
- Entity type 3 measures the functional size of failure analysis capability
- Entity relationship: many-many failures types

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and capability in the system

Entity type 4 (external measurement for analyzability)

- Entity name: status-monitoring capability
- Input of entity type 4: data monitor recording
- Output of entity type 4: actual failed data monitor
- Entity type 4 measures the functional size of status monitoring capability
- Entity relationship: many-many failures data monitoring in the system

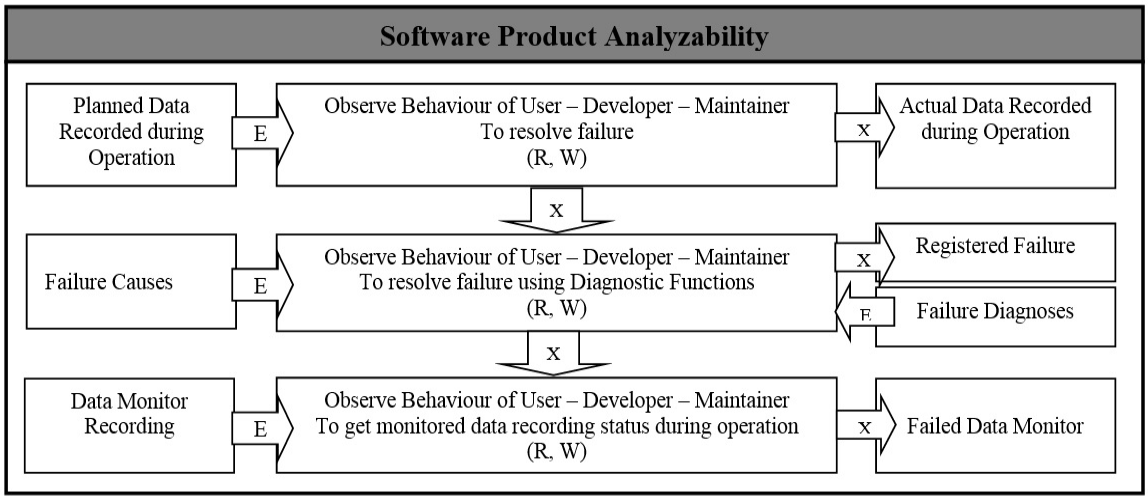

Figure 2 A metamodel of software product analyzability

4.3.2 Metamodel of Software Changeability Requirements

There are three entity types to capture the changeability requirements; change efficiency, modifiability, and software change control capability. Figure 3 presents a metamodel that represents the three identified entity types and their corresponding relationships. This metamodel represent the relationship between entity types in terms of input, process and output.

Entity Type 5 (external measurement for changeability)

- Entity name: change efficiency
- Input of entity type 5: planned time to change
- Output of entity type 5: actual work time to change
- Entity type 5 measures the functional size of change efficiency
- Entity relationship: many-many failures time to change in the system

Entity Type 6 (internal measurement for changeability)

- Entity name: modifiability
- Input of entity type 6: number cases of change software
- Output of the entity type 6: actual number cases of failing change
- Entity type 6 measures the functional size of modifiability
- Entity relationship: many-many number cases of change in the system

Entity Type 7 (external measurement for changeability)

- Entity name: software change control capability.
- Input of entity type 7: planned of change recorded of log data
- Output of entity type 7: change of log data actually recorded
- Entity type 7 measures the functional size of software change control capability
- Entity relationship: many-many of control change of log in the system

Journal of Theoretical and Applied Information Technology
30th June 2018. Vol.96. No 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

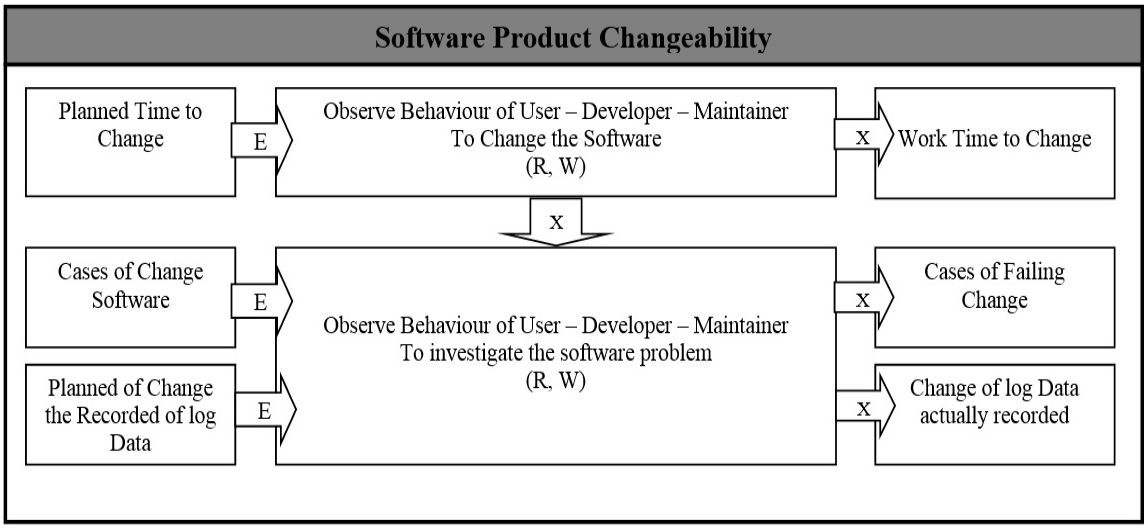

Figure 3 A metamodel of software product changeability

4.3.3 Metamodel of Software Stability Requirements

There are two entity types to capture the stability requirements; change success ratio, and modification impact. Figure 4 presents a metamodel that represents the three identified entity types and their corresponding relationships. This metamodel represent the relationship between entity types in terms of input, process and output.

Entity Type 8 (external measurement for stability)

- Entity name: change success ratio
- Input of entity type 8: software failure before change
- Output of entity type 8: software failure after change
- Entity type 8 measures the functional size of change success ratio
- Entity relationship: many-many of software change after/before in the system

Entity Type 9 (internal measurement for stability)

- Entity name: modification impact
- Input of entity type 9: planned of change after the first change of software
- Output of entity type 9: resolved failures
- Entity type 9 measures the functional size of modification impact
- Entity relationship: many-many of software change in the system

4.3.4 Metamodel of Software Testability Requirements

There are three entity types to capture the testability requirements; availability of built in test function, re-test efficiency, and test restartability. Figure 5 presents a metamodel that represents the three identified entity types and their corresponding relationships. This metamodel represent the relationship between entity types in terms of input, process and output.

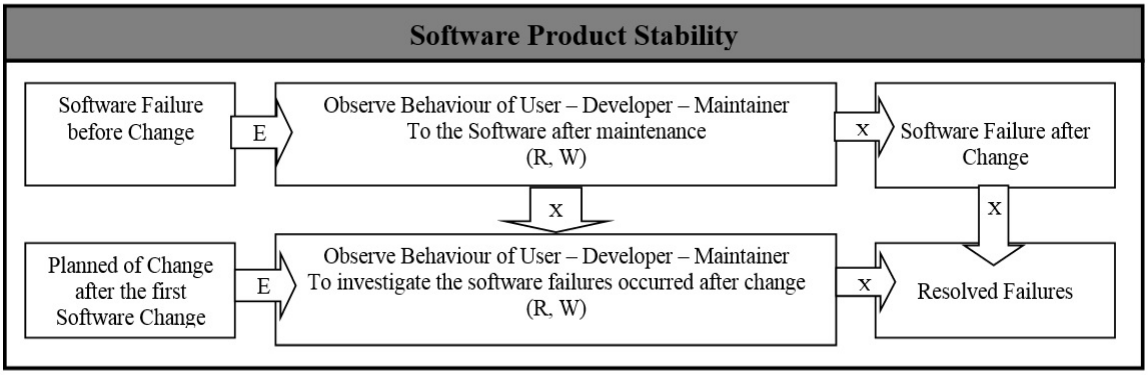

Figure 4 A metamodel of software product stability

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

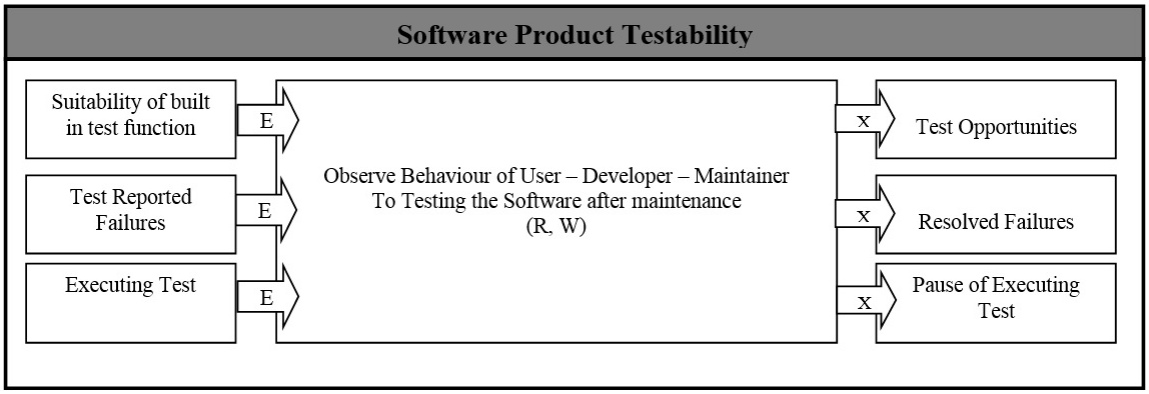

Figure 5 A metamodel of software product testability

Entity Type 10 (external measurement for testability)

- Entity name: availability of built in test function
- Input of entity type 10: suitability of built in test function
- Output of entity type 10: test opportunities
- Entity type 10 measures the functional size of availability of built in test function
- Entity relationship: many-many of functions test in the system

Entity Type 11 (external measurement for testability)

- Entity name: re-test efficiency
- Input of entity type 11: test reported failures
- Output of entity type 11: resolved failures
- Entity type 11 measures the functional size of the re-test efficiency
- Entity relationship: many-many of failure test in the system

Entity Type 12 (internal measurement for testability)

- Entity name: test restartability
- Input of entity type 12: executing test
- Output of entity type 12: pause of the executing test
- Entity type 12 measures the functional size of the test restartability
- Entity relationship: many-many of test executing in the system

4.4 Numerical Assignment Rules

The foundations of the numerical assignment rules for software maintainability requirements are presented in the previous metamodels of software product analyzability, changeability, stability and testability (See figures 2 to 5).

The numerical assignment rules can be described using descriptive text (i.e. practitioner description) or through mathematical expressions (i.e. formal theoretical viewpoint). For measurement purposes of software functional size, the international standard for software functional size measurement ISO19761 identifies the concept of a "functional process" as an elementary component of a set of functional user requirements; it includes a unique cohesive and independently executable set of data movement types.

As specified in ISO19761, the data movement types are Entry, eXit, Read, and Write. Each data movement type moves one data group type. Maintainability data groups form sources and/or to data destinations for software maintainability requirements. One (1) CFP (i.e., COSMIC Function Point) represent a functional size measurement of each counted data movement type.

Table 1 and 2 presents data sources/destinations of software maintainability requirements. In both tables, data sources/destinations of maintainability requirements are categorized in four categories, analyzability, changeability, stability, and testability (see column #1). Whereas, data sources/destinations are next presented in column #2 and finally the objects of interest are presented in column #3.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Data sources of software maintainability requirements

Categories	Data Sources	Objects of Interest
Analyzability	– Planned data recorded during operation – Actual data recorded during operation – Failure causes – Actual registered failure – Failure diagnoses – Data monitor recording – Actual failed data monitor	Data Data Access Failure Failure Data Data
Changeability	– Planned time to change – Actual work time to change – Number cases of change the software – Number cases of failing change – Planned of change the recorded of log data – Change of log data actually recorded	Time Time Cases of change Cases of change Data Data
Stability	– Software failure after change – Change success ratio – Software failure before change – Software failure after change	Failure Time ratio Failure Failure
Testability	– Suitability of built in test function – Test opportunities – Pause of the executing test – Test restartability – Test reported failures – Resolved failures	Function Test Time Time Failure Failure

Table 2: Data destinations of software maintainability requirements

Categories	Data Destinations
Analyzability	– Audit trial capability – Diagnostic function support – Failure analysis capability – Status monitoring capability
Changeability	– Change efficiency – Modifiability – Software change control capability
Stability	– Change success ratio – Modification impact
Testability	– Availability of built in test function – Re-test efficiency – Test restartability

5. QUALITY EVALUATION OF SOFTWARE MAINTAINABILITY

This section presents an extension of the proposed measurement model of maintainability requirements. Numerical assignments rules are built based on mathematical expressions using descriptive text rules in ISO25010 [18]. The numerical assignment rules are appended to the metamodels of analyzability, changeability, stability, and testability requirements. The resulting metamodels presented in this section represent instantiation metamodels of the proposed model. They can be used to identify and measure software resources requirements based on the concepts in ISO25010 (2011), which can be considered as quality evaluation of software maintainability requirements in addition to the measurement

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

benefit. Figures 6 to 9 presents instantiation metamodels to measure analyzability changeability, stability, and testability (externally/internally) of software product for one functional process.

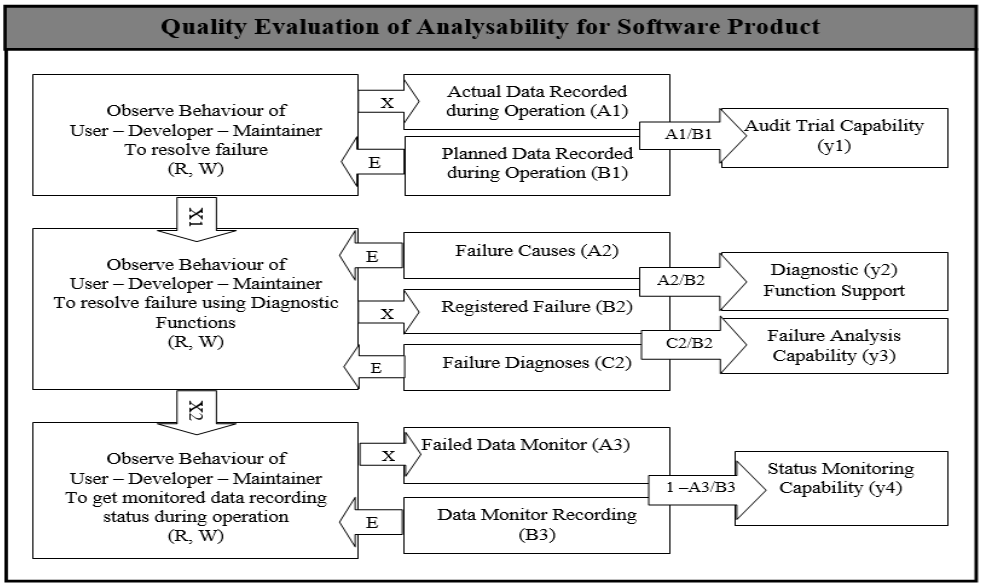

Figure 6 Quality evaluation metamodel of software product analyzability

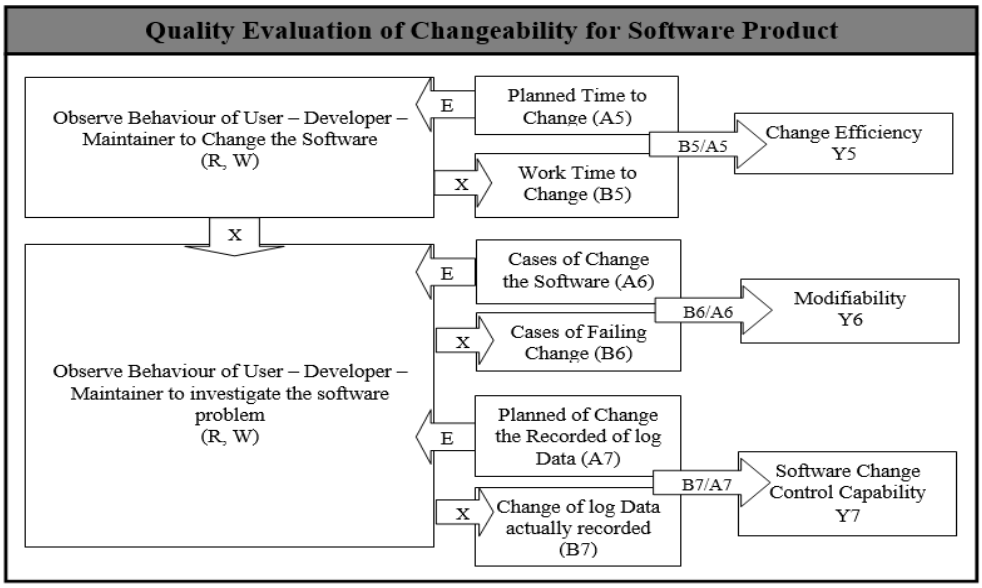

Figure 7 Quality evaluation metamodel of software product changeability

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

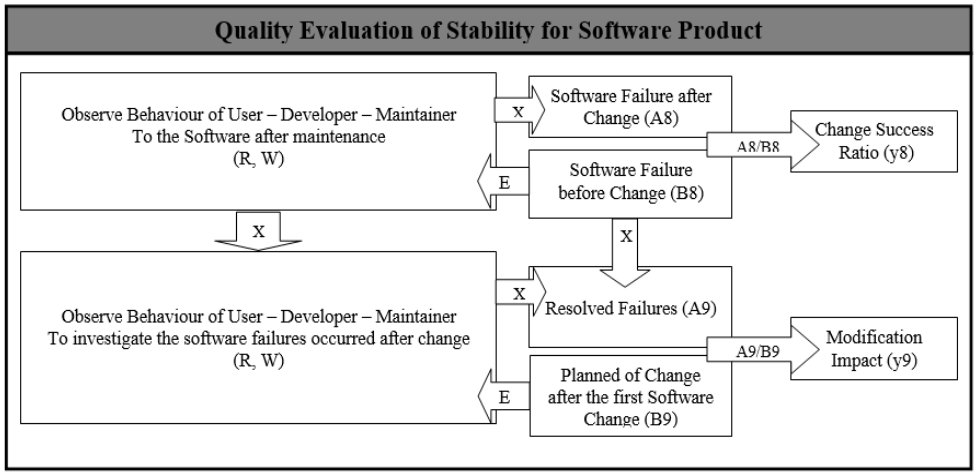

Figure 8 Quality evaluation metamodel of software product stability

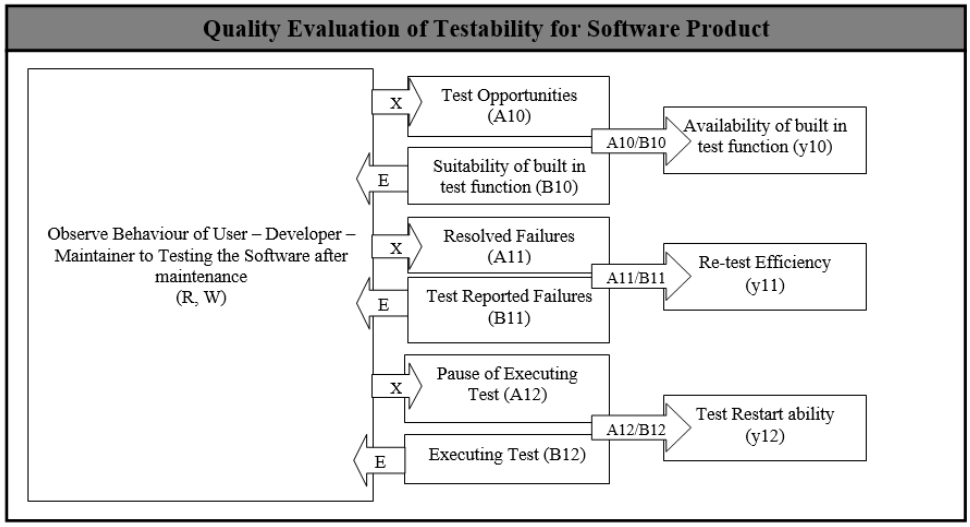

Figure 9 Quality evaluation metamodel of software product testability

6. DESIGN OF SOFTWARE STANDARD ETALON

Using a standard etalon can improve competitiveness by reducing the cost of both manufacturing and market transactions: a producer does not need to reinvent the specifications or performance criteria incorporated in the standard, and can therefore concentrate resources elsewhere. Furthermore, a standard etalon can contribute to the propagation of innovations, and consequently enhance the economic benefit to be derived from them [38]. With respect to the International Vocabulary of Basic and General Terms in Metrology a standard etalon is: *"A material measure, measuring instrument, reference material or measuring system intended to define, realizes, conserve or reproduce a unit or one or more values of a quantity to serve as a reference"* [38].

A system of references is made up of software measurement standards. Measurement standards are essential elements for an adequate metrological structure, in that they provide measurement users with a common reference and give them greater confidence in the measurement process. Indeed, standards facilitate the realization of measurement results on common basis.

In software engineering, concepts of units and etalons have seldom been used, and this is a symptom of the immaturity of the software measures themselves [38]. The measurement model presented in this paper can be considered as a reference model for measuring the functional size of the maintainability requirements for the following reasons:

- The definitions and the interpretation of the maintainability requirements are taken from the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

definitions of security requirements in the European international standard series (ECSS-E-40), IEEE-830 standard and ISO25010; this could be considered as a primary material measures to the proposed generic model of maintainability.

- The measurement model presented in this paper including four steps adopted from [38] these steps help to ensure that measurements are performed in a consistent manner; a base line is established as a primary reference.
- Using the ISO19761 standard as international method to measure functional size of maintainability requirements as well as provide measurement units.
- The calibration between steps (1), (2), and the COSMIC standard procedure to identify the proposed measurement model of maintainability requirements. This equivalent to a measurement instrument or reference material with respect of software etalon.

The proposed measurement model of maintainability requirements with respect to etalon standards offers:

1. The maintainability is measured internally and externally based on number of functional processes.
2. The proposed model provide measurement for each type or all types of maintainability requirements. For example, measurement for maintainability of analyzability, changeability, stability and testing.
3. The interrelations between internal and external measurements are defined.
4. The functional size measurements for software maintainability requirements are defined for all functional processes (externally and internally).
5. Using the proposed measurement model of maintainability requirements, the functional size measurement could be easily traced.
6. The proposed measurement model provide control and stability of the measurement results.
7. The proposed measurement model yield a measurement result with a standardized measurement unit (i.e. COSMIC function point).

Abran [38] designed a standard methodology to compare the design a software measurement standard Etalon with functional size measurement using ISO19761: COSMIC. This methodology adopt the proposed measurement model for the functional size measurement of the maintainability requirements as follows:

- Analysis and selection of candidate inputs for the maintainability requirements to begin the process of designing a standard etalon for maintainability requirements. In particular, it consists of the output of the definitions and interpretation of the ECSS European, ISO25010 and IEEE-830 standards as well as the identification of a set of candidate inputs for measurement.
- Identification of quality criteria of inputs (i.e. or the requirements). The quality criteria selected as prerequisites selected from the ECSS, ISO25010 and IEEE-830 standards of the maintainability requirements.
- Quality improvement of inputs. The input of the maintainability requirements therefore analyzed and improved using the quality criteria identified in the previous steps and consistency of the proposed for measuring the functional size of maintainability is based on the candidate inputs from the above standards.
- Selection or design of an etalon template to present measurement process and measurement results.

7. CASE STUDY: AN ONLINE LIBRARY SOFTWARE

7.1 Scope and Objective

This section presents a verification of the proposed measurement model using requirements specifications of an online library software. This software is developed to improve the services provided by a traditional library by providing an easy access to the information of books, journals and periodic publications. The requirements specifications used in this experiment are selected without a detailed inspection and analysis of their quality in terms of ambiguity and completeness. The reason for not conducting a quality inspection is to emulate the quality of software requirements specifications at an early phase of the software development life cycle. The objective of this experiment is to measure the functional size of maintainability requirements of an online library software using the proposed measurement model presented in this paper.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

7.2 Requirements Specifications of an Online Library Software

The online library software is developed to provide information about library resources including textbooks, journal, and periodic publications for its users (e.g. students, professors, librarians, and administrators). The users shall have an acceptable level of knowledge on computers and internet browsing. The library administrators shall have a good knowledge of the online software and be able to resolve typical issues that might arise and might be reported by the library users. Library resources are available on an online database in order to improve their accessibility for such users. The online library software shall use the host university information to provide the authentication service for its users. The library administrator grants the users in order to determine type of services they are entitled to use/book such as graduate studies study rooms. The credentials (i.e. username, password, and security verification questions) of the users can be changed via the user's portal. The users can access both internal and external databases to obtain access into a certain article or textbook. The users of the online library software can suggest to the library administrator buy an access for newly published resources. They shall have a usable interface that provide an easy access to library resources, user manual and online help to resolve issues during the library business hours. Further, the online library software shall be connected to the host university system to obtain necessary information to authenticate users trying to access resources from outside university campus. Therefore, the online library software shall be available 24 hours/day or schedule maintenance periods during users' inactive time using a specified time schedule. Figure 10 presents an overview of the context model of the online library software.

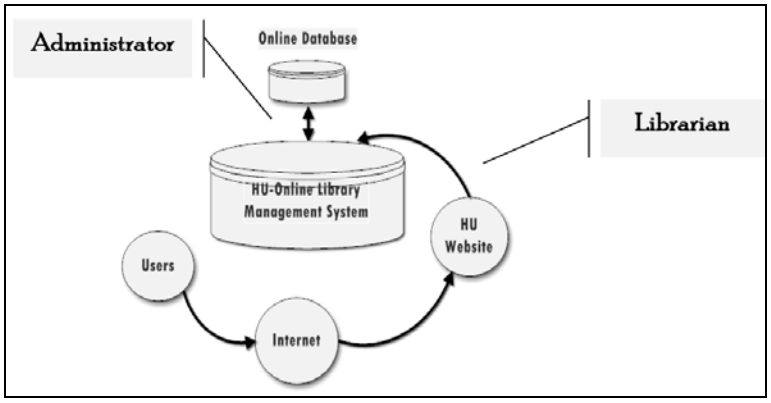

Figure 10 A context model of an online library software

7.3 Experimentation of the Proposed Measurement Model

This experiment measure the functional size of stability requirement of the online library software using the proposed measurement model. Using the requirements specifications presented in previous section, two functional processes can be identified using the proposed measurement model, namely "automatic change or update" and "modification impact" functional processes. For the "automatic change or update" functional process, three measures can be observed, as follows:

- Measure software failures after change
- Measure software failures before change by maintainer to software after maintenance
- Measure software failures after change by maintainer to software after maintenance

On the other hand, for the "modification impact" functional process, three measures can be observed, as follows:

- Measure planned of change after first software change
- Measure planned of change after first software change by maintainer to software failures occurred after change
- Measure software resolved failure by maintainer to software failures occurred after change

Table 3 presents the measurement of the functional size of stability requirement of the online library software using the proposed measurement model. In this table, column #1 presents the name

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of the identified functional processes. Column #2 presents the identification of the data movements exist in each identified functional process. Further, Column #3 presents the type of each identified data movement presented in column #2. For example, five data movements are identified in "automatic change or update" functional process, including one (1) Entry data movement data type, two (2) Read data movement types, and two (2) Write data movement types; and this yields a total of five (5) data movement types. These five (5) data movement types represent a functional size of five (5) COSMIC Function Points. The total functional size measurement of the two identified functional process is nine (9) COSMIC Function Points.

This measurement of the functional size of stability requirement forms one building block in the equation that measures the total functional size of maintainability requirement for the online library software. It is worth mentioning that such measure of maintainability requirement is beneficial as it is used in the estimation the effort required to maintain such software and more importantly at an early phase of the software development life cycle.

Table 3: Functional size measurement of stability requirement using the proposed measurement model

Functional Process	Data Movement Description	Data Movement Type
Automatic Change or update	o Entry software failures after change	(1) Entry
	o Read software failures before change by a maintainer to software after maintenance	(1) Read
	o Write software failures before change by a maintainer to software after maintenance	(1) Write
	o Read software failures after change by a maintainer to the software after maintenance	(1) Read
	o Write software failures after change by a maintainer to software after maintenance	(1) Write
Modification impact	o Read planned of change after first software change by a maintainer to the software failures occurred after change	(1) Read
	o Write planned of change after the first software change by a maintainer to the software failures occurred after change	(1) Write
	o Read software resolved failure by a maintainer to software failures occurred after change	(1) Read
	o Write software resolved failure by a maintainer to the software failures occurred after change	(1) Write
Total functional size of stability requirement		**9 CFP**

7.4 Threats to Validity

An Internal threat to validity might exist in the case of lack in the description of the concepts to be evaluated in this experiment. To mitigate the risk of having such threat to validity, the principal researcher who designed the measurement model has not experimented the proposed measurement model using the online library software. Another researcher (i.e. co-author) in addition to a pilot test has conducted this task to verify the validity of the experimental steps.

An external threat to validity might exist since the principal researcher and software engineering research community need to verify the experimental results can be generalized beyond the experimental settings. Therefore, to mitigate the risk of having such threat to validity, the proposed measurement model shall be experimented using requirements specifications that enables the measurement of maintainability requirement and not only stability requirement. Further, the proposed measurement model shall be experimented using requirements specifications that represents various software applications that works in different application domain.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

8. CONCLUSION

This paper presented the design of new measurement model of software maintainability requirements based on ISO international standards. The design of the measurement model specify a strategy of measurement rules to perform mapping with concepts of ISO19761 international standard. The motivation of this paper is to develop a measurement model that calculates the functional size of maintainability requirements at an early phase of the software development life cycle.

Quality evaluation metamodels are also proposed in this paper; these metamodels identify and measure maintainability requirements of a software product using the concepts exist in ISO25010 [17] systems and software quality requirements and evaluation (SQuaRE) international standard. Furthermore, this paper presented the design of a software standard etalon.

An experiment is conducted to verify the applicability of the proposed measurement model using requirements specifications of an online library software. This experiment measured the functional size of stability requirement as part of measurement for maintainability requirement of such software. Future work shall be devoted to conduct more experimentation of the proposed measurement model for applicability, and to mitigate the impact of external threat to validity using software requirements specifications that represent different software applications.

REFERENCES

[1] S. H. Kan, Metrics and Models in Software Quality Engineering, second edition, Addison-Wesley Longman Publishing Co, Boston, USA, 2002.

[2] K. Almakadmeh, K. Meridji, K. T. Al-Sarayreh, "Towards a reference model of software resources quality", *Journal of Computer Science*, Vol. 14, No. 2, pp. 182-198.

[3] V. Podgorelec, and M. Heričko, "Estimating software complexity from UML models", *ACM SIGSOFT Software Engineering Notes*, Vol. 32, No. 2, 2007, pp. 1-5.

[4] A. Idri, A. Abran, and T.M. Khoshgoftaar, "Evaluating software project effort by analogy based on linguistic values", *Eighth international software metrics symposium*, Ottawa, Canada, 2002, pp. 21-30.

[5] P. Bourque, "Estimating effort and cost in software projects - ISBSG a multi-organizational project data repository for project estimation and benchmarking", University of Ulster, Northern Ireland, 2003.

[6] R. Dawson and B. O'Neill, "Simple metrics for improving software process performance and capability: a case study", *Software Quality Journal*, Vol. 11, No. 3, 2003, pp. 243-258.

[7] P. Kokol, J. Brest, "Software complexity metric with the critical value", *International Conference on Computational Cybernetics and Simulation Systems, Man, and Cybernetics*, Orlando, USA, 1997, pp. 494-499.

[8] E.H. Alikacem, and H. Sahraoui, "A metric extraction framework based on a high-level description language", *Proceedings of the ninth international working conference on source code analysis and manipulation*, Edmonton, Canada, 2009, pp. 159-167.

[9] R. Dawson, A.J. Nolan, "Towards a successful software metrics programme", *Eleventh annual international workshop on software technology and engineering practice*, Amsterdam, Netherlands, 2003, pp. 48-51.

[10] European Cooperation for Space Standardization, "Space Engineering: Software - Part 1 Principles and Requirements (ECSS-E-40-Part-1B)", European Cooperation for Space Standardization, Noordwijk, Netherlands, 2003.

[11] European Cooperation for Space Standardization, "Space Engineering: Software - part 2 Document Requirements Definitions (ECSS-E-40-Part-2B)", European Cooperation for Space Standardization, Noordwijk, Netherlands, 2005.

[12] European Cooperation for Space Standardization, "Space product assurance: software product assurance (ECSS-Q-80B)", European Cooperation for Space Standardization, Noordwijk, Netherlands, 2003.

[13] International Organization for Standardization, "Systems and software engineering - software life cycle processes", International Organization for Standardization, Geneva, Switzerland, 2008.

[14] European Cooperation for Space Standardization, "Space product assurance: dependability (ECSS-Q-ST-30C)", European Cooperation for Space Standardization, Noordwijk, Netherlands, 2009.

[15] International Organization for Standardization, "Systems and software engineering – vocabulary", International Organization for Standardization, Geneva, Switzerland, 2010.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[16] Institute of Electrical and Electronics Engineers, "International standard for software engineering - software life cycle processes – Maintenance (IEEE 14764)", IEEE Computer Society Press. USA, 2006

[17] Institute of Electrical and Electronics Engineers, "IEEE standard dictionary of measures of the software aspects of dependability (IEEE982.1)", IEEE Computer Society Press. USA, 2005.

[18] International standardization organization, ISO25010, "Systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - System and software quality models", International organization for standardization, Geneva, Switzerland, 2011

[19] International Organization for Standardization, "Information Technology - software measurement - functional size measurement Part 1: definition of concepts", International Organization for Standardization, Geneva, Switzerland, 2007.

[20] International Organization for Standardization "ISO19761: A functional size measurement method: COSMIC", International organization for standardization, Geneva, Switzerland, 2013.

[21] International Organization for Standardization, "Software Engineering - NESMA functional size measurement method (ISO/IEC 24570) - definitions and counting guidelines for the application of function point analysis", International organization for standardization, Geneva, Switzerland, 2005.

[22] P. Forselius, "Faster and more accurate functional size measurement by KISS - keeping it simple", *IFPUG metric views*, Cambridge, USA, 2006, pp. 1-10.

[23] D. Port, and B. Taber, "Actionable analytics for strategic maintenance of critical software: an industry experience report", *IEEE Software*, Vol. 35, No. 1, 2017, pp. 58-63.

[24] H. Wu, L. Shi, C. Chen, Q. Wang, and B. Boehm, "*Maintenance effort estimation for open source software: a systematic literature review*", IEEE International Conference on Software Maintenance and Evolution, Raleigh, NC, USA, 2016, pp. 32-43.

[25] C.-J. Lin, and D.-M. Yeh, "*A software maintenance project size estimation tool based on COSMIC full function point*", International Computer Symposium, Chiayi, Taiwan, 2016, pp. 555-560.

[26] R. Torkhan, J. Laval, M. Derras, and N. Moalla, "*Conceptual interoperability framework for software development and maintenance*", International Conference on Engineering, Technology and Innovation, Funchal, Portugal, 2017, pp. 1327-1332.

[27] N. Al-Saiyd, "*Source code comprehension analysis in software maintenance*", 2nd International Conference on Computer and Communication Systems, Krakow, Poland, 2017, pp. 1-5.

[28] D. Manev, and A. Dimov, "*Facilitation of IoT software maintenance via code analysis and generation*", 2nd International Multidisciplinary Conference on Computer and Energy Science, Split, Croatia, 2017, pp. 1-6.

[29] M. Yan, X Xia, X. Zhang, D. Yang, and L. Xu, "*Automating aggregation for software quality modeling*", IEEE International Conference on Software Maintenance and Evolution, Shanghai, China, 2017, pp. 529-533.

[30] J. Alhilman, F. Atmaji, and N. Athari, "*Software application for maintenance system: a combination of maintenance methods in printing industry*", 5th International Conference on Information and Communication Technology, Malacca City, Malaysia, 2017, pp. 1-6.

[31] M. Gupta, "*Improving software maintenance using process mining and predictive analytics*", IEEE International Conference on Software Maintenance and Evolution, Shanghai, China, 2017, pp. 681-686.

[32] R. Malhotra, and A. Chug, "An empirical study to assess the effects of refactoring on software maintainability", *International Conference on Advances in Computing, Communications and Informatics*, Jaipur, India, 2016, pp. 110-117.

[33] N. Mellegard, A. Ferwerda, K. Lind, R. Heldal, and M. Chaudron, "Impact of introducing domain-specific modelling in software maintenance: an industrial case study", *IEEE Transactions on Software Engineering*, Vol. 42, No. 3, 2016, pp. 248-263.

[34] R. Plösch, S. Schürz, and C. Körner, "On the validity of the IT-CISQ quality model for automatic measurement of maintainability", *IEEE 39th annual international computers, software & applications conference*, Taichung, Taiwan, 2015, pp. 326-334.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[35] G. Szőke, C. Nagy, P. Hegedűs, R. Ferenc, and T. Gyimóthy, "Do automatic refactorings improve maintainability? An industrial case study", *IEEE International Conference on Software Maintenance and Evolution*, Bremen, Germany, 2015, pp. 429-438.

[36] T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, "A probabilistic software quality model", *Proceedings of 27th IEEE international conference on software maintenance*, Williamsburg, USA, 2011, pp. 243–252.

[37] S. Counsell, X. Liu, S. Eldh, R. Tonelli, M. Marchesi, G. Concas, and A. Murgia, "Re-visiting the maintainability index metric from an object-oriented perspective", *41st Euromicro Conference on software engineering and advanced applications*, Funchal, Portugal, 2015, pp. 84-87.

[38] Alain Abran, 2010. Software Metrics and Software Metrology, IEEE Computer Society Press.