Journal of Engineering and Applied Sciences 15 (1): 252-260, 2020
ISSN: 1816-949X

Formal Framework for Semi-Automation of Validation of Software and its Metrics and of Development of Relevant Knowledge-Base

Meenakshi Sridhar and Nasib Singh Gill
Department of Computer Science and Applications, M.D. University, Rohtak, India

Abstract: The task of validation of software development is quite complex and crucial one for developing appropriate, efficient and robust software. The complexities in development of software and of its validation arise because of various factors including the large number of stakeholders, types of software and applications, paradigms and models and methodologies and frameworks employed for its development. Also, for proper validation, software characteristics viz. functionality, reliability, robustness and integrity need to be considered. In view of the fact of the task being quite complex, it is desirable that processes for validation be systematic and be automated as far as possible. Also, validation is directly related to requirements, some of which may change even till the final delivery. Thus, the proposed framework should be desirably flexible. In this communication, a formal framework is proposed which semi-automates the task of validation, etc. by helping in locating or proposing an appropriate validation method or procedure and the framework is flexible enough to incorporate changes with minimum efforts. The framework assumes LISP-like environment and uses single Abstract Data Type (ADT), viz. the list (or recursive list). The ADT is also basic program construct having the advantage that even programs may be treated as data and can be given as inputs including for modification, to any other program or to itself.

Key words: Formal methods in software engineering, automating validation, validation metrics, procedure, program, framework

INTRODUCTION

Empirical validation, whether of software metrics or for any other practical software concern is quite a complex task in view of various factors including large number of types of software, number of required characteristics of software, number of paradigms and models and of methodologies and frameworks used. In order to manage the complexity, it is highly desirable that the task of validation be partially or fully automated.

In this respect, it may be clarified in the beginning itself that human capabilities/intelligence and machine capabilities/intelligence are in many respects, complimentary and appropriate choices from the two at different steps/stages may facilitate design and development of software solutions. The human intelligence is rooted in judgmental evaluation, commonsense, informal expression and inductive reasoning. On the other hand, the machine intelligence is essentially based on formal expression, deductive reasoning and formal rule-based evaluation, etc. Of the two, so far, the human intelligence is essentially required for understanding the problem domains, their environments and for design part of software solution. Also, human intelligence is needed for developing solutions in such a form that it is understood and executed by the machine.

Why only semi-automation and not full automation? The pioneering theoretical computer science work, during 1930's by young brilliant mathematicians including Turing, Church, Godel and post proved that algorithmic/ computational approach is insufficient for solving most of the difficult problems which human beings encounter. More explicitly, it was proved that even the most advanced computer which will be available at any point of time in future will be able to solve by itself and without human intervention, only a very small fraction of the problems encountered ever by humanity.

Thus, the pioneering work established long before the first ever stored program computer made its first calculation the need for essential involvement of human intelligence for solving most of the difficult problems.

Later on, the pioneers including Brooks (1986), from the field of software engineering discovered, through practical experience of developing software, that for

Corresponding Author: Meenakshi Sridhar, Department of Computer Science and Applications, M.D. University, Rohtak, India

solving complex problems the machine intelligence is useful mainly in the accidental repetitive tasks of problem solving, the tasks which could be automated. These software experts discovered and emphasized the indispensability of human intelligence, specially, human judgment, creativity and expertise for handling that part of problem solving which they called the 'Essential tasks'.

Later, even the psychologists and others including Mair and Shepperd (2011) have re-emphasized that the human intelligence is indispensible for many problem solving tasks including the ones for designing software. From the above discussion, it is irrefutably established that man-machine combination is essential for attempting solutions of difficult problems.

Even for those problems which can be fully automated, a fully automated solution for some of such problems may have exponential complexity. However, complexities of solutions of such problems may be reduced to polynomial ones through human intervention at appropriate points.

Foundational concepts, issues and explanations: As validation is the core concept for this research, the discussion in this study is initiated with a working definition for it. In brief, validation is the process of ensuring that the right product is built where 'Right product' means it meets the needs and goals of various stakeholders of the final software product. The needs and goals of the various stakeholders are collected in the form of requirement specification which is desirably defined before the process of software development is initiated. However, the process of defining requirement specification is a continual one which may continue even almost till the final product is delivered.

Another concept, viz. verification of software, though closely related is yet a distinct concept. The distinction between the two may be clarified by stating that Validation is about ensuring finally right product is built, whereas, verification is about ensuring that the process of building the product be right. In a systematic phased development of software, verification is conducted in phased manner after each phase of development. Each phase is given relevant specification as input. Verification, at the conclusion of the phase, ensures that the output of the phase meets the input specification.

Next, some facts which need to be taken into consideration while developing semi-automated solution for validation of software and relevant metrics are discussed.

A number of paradigms and models which include waterfall, incremental, prototyping, spiral, cleanroom, agile, etc. are available for developing software. A number of methodologies and frameworks which include Scrum, V-Model, RAD, XP, Lean, DevOps, FDD, MDD, DSDM, etc. are used for developing software.

The choices of appropriate paradigm/model and of methodology/framework depend on the type of problem and its domain for which some software solution is required. Conversely, the choices determine how the required software is developed. The required characteristics of software which include functionality, reliability, usability, efficiency, maintainability, portability, robustness and integrity need to be kept in mind and to be ensured throughout the development of software. The number of types of software is quite large depending upon various factors including intended usage and types of problem domains.

As is required for the purpose, next, types of software are considered in some details. Each of the following classifications covers most of the available software types but from a different perspective.

Classification 1; Based on common function, type or field use of software: System software which provides platform for running application software and which directly operates the computer hardware, application software designed to help users to perform specific tasks. Computer programming tools such as compilers and linkers which are used to translate and combine computer program source code and libraries into executable code other software tools which help in designing and developing other software.

Classification 2: Software based on type of software development and engineering involved: Software engineering, web engineering for designing software involving dominantly hypertext/hypermedia and cloud engineering for designing software for which the configurable resources like computer networks, servers, storage, applications and services are assumed to be provided and managed by third party, embedded software engineering for developing software for controlling processes of various devices and machines which are different from computing devices. For example, aircraft carrier guidance software, mobile software engineering, API development concerned with developing tools which allow software functioning across different operating systems on desktop, mobile and the web, software tools development concerned with developing tools which

allow other software developers to test their code, so as to ensure their code conforms to industry standards and is maintainable.

Classification 3; Microsoft TechNet and AIS Software classification has seven major elements: Platform and management, education and reference, home and entertainment, content and communication, operations and professional, product manufacturing and service delivery and line of business. Another possible categorization of software is based on copyright status.

Classification 4; Classes in software based on copyright status: Classification include Free Software, Open Source Software, Copylefted Software, Non-copylefted Free Software, Shareware, Freeware, proprietary, etc.

Each of the types within a classification scheme can itself be considered as consisting of (sub) classes. For example, System software (software which is primarily used to operate the hardware) may be further classified as time sharing, resource sharing, client server, batch processing operating system, real time operating system, multi-processing operating system, multi-programming operating system and distributed operating system. This classification of system software may be called, say, Scheme 1-System-Software. Also, the System Software may be classified as operating systems, utility programs, library programs, language translators (assembler, compiler and interpreter). This classification of System Software may be called, say, Scheme 2-System-Software.

Literature review: Earliest available literature on validation (Boehm, 1976, 1984; Briand *et al*., 1995; Balci, 1998) deals with general issues including guidelines and principles of software validation. Some later similar literature includes (Steven, 2001; Pohl and Rupp, 2010).

Earliest specific literature (Schneidewind, 1994; Bell and Brat, 2008; Marculescu, 2010; Anonymous, 2002) is about Space Exploration Software. Literature regarding Validation of component-oriented and object-oriented software systems and metrics includes, respectively (Dolado, 2000; Skroch, 2007; Tomar and Gill, 2010; Szabo and Teo, 2012; Khan *et al*., 2013; Alshayeb and Li, 2003; Bajeh *et al*., 2014). Validation of Information security is covered by Fenz and Ekelhart (2011) and Michael *et al*. (2010). The topic of validation of software metrics, in general is treated by Bajeh *et al*. (2014), Cruickshank *et al*. (2009), Michael *et al*. (2010), Canfora *et al*. (2005), Oberkampf and Barone (2006), Liu *et al*. (2011), Subramani *et al*. (2014), Veerappa and Harrison (2013), Jamil *et al*. (2010), Xargay *et al*. (2009), Venkitachalam *et al*. (2015), Ramirez *et al*. (2010), Tripathi *et al*. (2008) and Li (2016). Further, Liu *et al*. (2011), Kaivola *et al*. (2009), Ando *et al*. (2015), Yan *et al*. (2015), Bagalini and Violante (2016), Csertan *et al*. (2002) and Koopman (2015) discuss formal models and issues related to automation of testing and validation. Validation of some other specific software including those for embedded systems, online games and pharmaceutical applications are discussed, respectively by Koopman (2015), Usai *et al*. (2017) and Johanning *et al*. (2014).

MATERIALS AND METHODS

Detailed outline of the proposed approach: In this study, apart from the detailed outline of the proposed approach, classification of application software, used for exemplars in the next study is discussed in detail.

Outline of the proposed approach and its advantages: The proposed approach has the following major aspects; According to the proposed approach, the various activities for semi-automation including those for design of validation task are initially expressed in some informal natural language. The natural language expressions are then translated to semi-formal/ mathematical entities in the form of recursive lists. The translation of recursive lists, so, obtained to corresponding formal expressions of either a functional language like LISP or of a logical programming language like PROLOG is almost straight forward.

The proposed approach is top-down, distributed and flexible. The Top-down aspect of the proposed approach may be sketched as follows: it may be recalled that the task of validation is regarding software. To begin with, the totality of software may be considered as a three-element list: (Totality-of-software, classification scheme, list-of classes-in-scheme). For example, (Totality-of-software, classification 1, (System Software, Application Software, computer programming tools, other software tools)).

Next, each of the components, say System Software, can be further expanded as a list. For example, (System Software, Scheme 2-System-Software, (Operating systems, utility programs, library programs, language translators). Further, language translators may be further expanded as a list: (Language translators, xxx, (Assembler, compiler and interpreter)) where, ‘xxx’ may be replaced by some classification of language translators. However, if there is only one classification, ‘xxx’ may be kept blank.

The process may be continued as long as some element in the third component of the list of the previous step can still be decomposed, e.g., (compiler, classification-5, (Cross-compiler, source-to-source compiler, Incremental-compiler, Source-compiler).

The approach is used not only for classification into subclasses. It can also be used for stating other aspects

like components or properties. For example, compiler may be further defined in terms of its functions as (compiler, phases-of-compiler, (lexical-analysis, syntax-analysis, semantic-analysis)) where, phases-of-compiler denotes some aspect/characteristics of compiler.

Further, the approach is flexible in the sense that automation process is not necessarily strictly top-down. For example, we may define (compiler, phases-of-compiler, (lexical-analysis, syntax-analysis, semantic-analysis, …) even before defining language-translators for which it is to be a part of its formal definition or even before defining System Software. Later on, when the term 'Language translators' is formally defined as a recursive list, at that time the definition of 'compiler' may be inserted as an element in the recursive list defining 'Language translators'. Also, if we have already defined System Software formally in which up to a particular point of time, 'Language translators' was not considered as one of its required constituent/attribute. But suppose later, it is found that 'Language-translators' is also essentially required as constituent/attribute. The approach allows the required modification easily.

The task of developing a formal model of the problem domain and its environment (Sridhar and Gill, 2015a, b; Serban *et al*., 2010; Goulao and Abreu, 2005; Woodcock *et al*., 2009 and Zhixiong *et al*., 2007) is among the first ones for automation of a task. The task of formalization is composed of two major parts: giving formal definitions of the structure of problem domain, its environment and interface between the two and giving formal definitions of the dynamics within each of the problem domain, the interface and some relevant aspects of the environment. The structure of a system consists of elements and components of the system, relations and even meta-relations between the elements/components. The dynamics with in a system refers to various processes that go inside the system/interface which either transform one state to another state of system while maintaining its equilibrium or transform the very structure of the system/interface.

For the structure, a broad 4-step outline of iterative method for formalizing each of domain, interface and environment is given:

Step 1: The formalizing process starts with denoting formally some most fundamental concepts of the problem domain and later of the part of environment of the problem domain and of interfaces between the two.

Step 2: Next, properties of the denoted concepts, relations and meta-relations, etc., between these concepts are formally defined and denoted. Defining/denoting formally means expressing these in terms of mathematical sets, relations and functions, etc.

Step 3: Next, consider some (maximum up to, say, 7) new concepts directly definable from the fundamental concepts and then formally define the new concepts in terms of the already defined concepts.

Step 4: Treat some of the concepts obtained in the previous step as fundamental concepts, go to step 2. Similarly, the dynamics with in a system may be formally defined by replacing in steps 1-4 above, the term 'Concept' by the term 'Process'. The method of formalization delineated above has the following advantages.

The 4-step iterative process discussed above may be easily implemented in any functional language like LISP or a logical programming language like PROLOG. A term, once defined, may be easily expanded/modified later. According to the proposed framework, a conceptual framework may be easily expanded both upwards as well as downwards. For example, let language-translators be the first to be defined as (Language translators, xxx, (Assembler, compiler, interpreter)) without having defined System Software and compiler. Later on, let, System Software may be defined with language-translators as one of its components and also, compiler may be defined as (compiler, classification-5, (Cross-compiler, source-to-source-compiler, Incremental-compiler, source compiler) with compiler being a component of language translators. The approach allows writing of the code for each of language-translators, system software and compiler independently of the others.

Classification of application software: The ideas explained above are applied, in the next study for semi-automating the task of validating software, of relevant metrics and relevant knowledge-base. The explanation is mainly in respect of Application Software, the most dominant class of software under classification. For the purpose, some subclasses of Application Software are enumerated (in alphabetic order) as: Assistive Software for differently enabled, Automation Software, Automotive Software for monitoring, guiding and control of vehicles, Business and Financial Software, Communication Software, Design Software (CAD, CAM, robotics), Data and Knowledge base Software, Educational, Scientific, Engineering and services Software, Embedded Application Software, Emergency, Disaster Forecasting and Management

Information Software, Entertainment Software, Exploratory Software (space, minerals, oil), GIS, Governance Software, Healthcare Software, Manufacturing Software (3D Printing), Military Software for Missile Control, Monitoring Software, Searching Software (Search engines), Virtuality and Simulation Software.

Most of the above mentioned subclasses with in Application Software class may be further classified based on service/function provided. For example, Business and Financial Software may be further classified as: Business Software (Payroll calculations, budgeting, sales analysis, financial forecasting, managing employee database, maintenance of stocks, etc.) and Banking Software (Online accounting facility, ATM machines), marketing: advertising and home shopping.

Similarly, Healthcare may be further classified as diagnostic system for collecting data and for identifying cause of illness, lab-diagnostic system to conduct tests and to prepare reports, patient monitoring system to check the patient's signs for abnormality as in Cardiac Arrest, ECG, etc., pharma information system to check drug labels, expiry dates, harmful side effects, etc., surgery for performing surgery.

Also, design (engineering) may be further classified as structural engineering for say, analyzing stress and strain in respect of design of ships, buildings, budgets, airplanes, etc., industrial engineering for dealing with design, implementation and improvement of integrated systems of people, materials and equipment and architectural engineering for helping in planning towns, designing buildings, determining a range of buildings on a site using both 2D and 3D drawings. Next, the above classification along with preceding outline is used to illustrate the approach.

RESULTS AND DISCUSSION

Exemplars explaining the proposed approach: In this study, the approach to the proposed framework is discussed in sufficient details through a number of exemplars.

Formal notation and semi-automation of knowledge-base for software and validation: For developing computational framework (instead of the mathematical one of section for the proposed semi-automated solution, the notation used in the previous study for denoting various concepts like totality of software, etc., have to be slightly modified in the manner explained. For example, the representation of totality of software as a three-element list: (Totality-of-software, Classification 1, (System Software, Application Software, computer programming tools, other software tools)) is slightly modified by incorporating (sub-discipline-path (Totality-of-software)), so that, the new notation becomes (Totality-of-software, (sub-discipline-path (Totality-of-software)) (Classification 1, (System Software, Application Software, computer programming tools, other software tools))).

The highlighted portion indicates the inserted modification. The purpose of the modification is to keep track of the lowest level of sub-discipline at which a validation metric may be defined/introduced or a validation procedure may be initiated from its code. The purpose will get more clarified as the development of the proposed framework is explained in increasing details. To facilitate human understanding, the notation is rewritten as:

Algorithm 1; Totality-of-software:

```
(Totality-of-software,
      (sub-discipline-path (Totality-of-software) )
   (Classification 1,
      (System software, Application software, Computer programming tools, Other software tools)
   )
)
```

Next, in order to explain how the notation is further expanded, let it be assumed that the System Software, according to Scheme 2-System-software may be classified as Operating Systems, Utility Programs, Library Programs, Language Translators). Then the expanded notation may be in either of the following two forms:

Algorithm 2; Totality-of-software:

```
(Totality-of-software,
      (sub-discipline-path (Totality-of-software) )
    (Classification1, (System software, Application software, Computer programming tools, Other software tools)))

      (System Software, (sub-discipline-path (Totality-of-software, System software))
                  (Scheme2-System-software, (Operating Systems, Utility Programs, Library Programs,
                        Language Translators))
```

Or

Algotrithm 3; Totality-of-software:

```
(Totality-of-software,
      (sub-discipline-path (Totality-of-software) )
      (Classification1, (
                  (System Software, (sub-discipline-path (Totality-of-software, System software))
                  (Scheme2-System-software, (Operating Systems, Utility Programs, Library
                        Programs, Language Translators)
                  ),
```

However, for ease of human understanding and ease of incorporating modifications, only the Algorithm 2 will be used in the rest of the communication.

Flexible and distributive nature of approach for concurrent development: Next, in order to demonstrate the flexible and distributive nature of the approach, let it be assumed that instead of 'System Software' classification, 'Application Software' becomes available as:

> (Application Software, Scheme 1-Application Software, (business-software, scientific Software, Healthcare-Software, Design Software) Instead of algorithm 2, the following becomes the required notation

Algorithm 4; Totality-of-software:

```
(Totality-of-software,
      (sub-discipline-path (Totality-of-software) )
     (Classification1, (System software, Application software, Computer programming tools, Other software tools)))

       (Application-Software, (sub-discipline-path (Totality-of-software, Application-software) )
            (Scheme1-Application-software, (business-software, scientific software, Healthcare-software,
                  Design-software, …)
      )
```

Now, if the system-software classification becomes available, the notation algorithm 4 can be extended as:

Algorithm 5; Totality-of-software:

```
(Totality-of-software,
      (sub-discipline-path (Totality-of-software) )
     (Classification1 (System software, Application software, Computer programming tools, Other software tools)))
       (Application-Software, (sub-discipline-path (Totality-of-software, Application-software) )
            (Scheme 1-Application-software, (business-software, scientific-software, Healthcare-software,
                  Design-software, …)
)
      (System Software, (sub-discipline-path (Totality-of-software, System software))
                        (Scheme2-System-software, (Operating Systems, Utility Programs, Library Programs,
                        Language Translators))
```

The same notation algorithm 5 will be obtained, if the order of availability of classifications is first for System Software followed by that for Application-Software. Thus, the framework is flexible as well as distributive. Also, if the software to be developed is very large as would be this one, if fully developed, different teams may develop parts of software independent of each other, e.g., one may develop System-Software part and the other the Application Software part and then merge the two. Thus, the development of software through the proposed framework is concurrent also.

Next, if classification of health-care becomes available as data collecting system, lab-diagnostic system, patient monitoring system, pharma information system and surgery, then complete notation, so far can be obtained by appending the following code to algorithm 5above as

> (Health-care, (sub-discipline-path (Totality-of software, Application-Software, Health-care)) (nil, (Data collecting system, lab-diagnostic system, patient monitoring system, pharma information system and surgery) where 'nil' indicates that no further classification is required

Characterization of validation metrics in the framework for semi-automation: So far, the approach for formalization and semi-automation of relevant knowledge is illustrated only in respect of a sort of hierarchy of disciplines and sub-disciplines of software, for which validation process may be required to be applied. However, during the process of extending definitions, a stage reaches when relevant to a particular discipline/ sub-discipline, definition/algorithm of a particular validation metric ultimately needs to be given and inserted in the framework. A metric, in the proposed framework, may be characterized as a recursive list in the following manner:

> ((attribute-name 1, attribute-value1) (attribute name 2, attribute-value 2), ... (attribute-namek, attribute-valuek), …) where some of the attribute-values may be links-addresses

For example, Computational Framework for a semi-automated software validation metric may be of the form:

> ((name-of-validation-metric, ???) (sub-discipline path (Totality-of-software, top-sub-discipline, next-level-sub-discipline,...)) validation-metric), (definition-or-algorithm-of validation-metric, ???)(link-to-executable-code-of-validation-metric, ???))

For the purpose of presenting some concrete exemplars, let it be assumed that the Application-software consists of four classes from the set {space-application, object-oriented, component-based, embedded}. Then its notation in the framework is (Application-software, sub-discipline-path (totality-of-software, Application software) (classification, (space-application, object

oriented, component-based, embedded)) Further, let space-application be classified by classification-1 as (a-s-1, a-s-2, a-s-3, …).

Then, notation becomes (Application-software, sub-discipline-path (totality-of-software, Application software) (classification, (space-application, object oriented, component-based, embedded)). (space application, sub-discipline-path (totality-of-software, Application-software, space-application) (classification-1, (a-s-1, a-s-2, a-s-3, …)).

Now, assume that there is no further classifications of any of a-s-1, a-s-2, a-s-3, ..., then, there may be various issues associated with each a-s-1, a-s-2, a-s-3 of a class a-s-i, say, its definition, validation metrics etc.

The previous notation may be further expanded as (a-s-i, sub-discipline-path (totality-of-software, Application-software, space-application, a-s-i) (definition-of-subclass, …, validation-metrics-of-subclass (link-to-set-of-definition-of-validation-metrics, set-of links-to-validation-metrics-code)). In summary, the approach allows development of the intended system in small steps concurrently.

CONCLUSION

Validation, along with appropriate procedures and metrics for it, plays significant role in developing appropriate, efficient and robust software. There is a large number of types of software with each type requiring specific validation procedures and metrics. It is desirable to create a semi-automated framework for knowledge base of validation metrics which allows quick access to the appropriate metrics and which is flexible enough to allow modifications when required. A framework is proposed for the purpose based on LISP-like notation, using which, a knowledge base consisting of definitions, tools, techniques, algorithms, executable codes etc. may be created for semi-automating the task of software validation.

The explanation in the previous two sections outlines how through the proposed framework an appropriate validation metric is located, details about the validation metric can be found and the validation process may be applied. The total task of semi-automating software validation, in general, is quite large which may require hundreds of man-hours but the foundational computational structure has been explained above in sufficient detail.

REFERENCES

Alshayeb, M. and W. Li, 2003. An empirical validation of object-oriented metrics in two different iterative software processes. IEEE Trans. Software Eng., 29: 1043-1049.

Ando, T., H. Yatsu, K. Hisazumi, A. Fukuda and M. Matsumoto *et al*., 2015. Reference model of specifications toward independent verification and validation. Proceedings of the IEEE Region 10th Conference on TENCON 2015-2015, November 1-4, 2015, IEEE, Macao, China, ISBN:978-1-4799-8639-2, pp: 1-3.

Anonymous, 2002. Department of Health and Human Services. Food and Drug Administration. Center for Devices and Radiological Health, USA. https://www.fda.gov/OHRMS/DOCKETS/98fr/02n-0276-npr0001.pdf.

Bagalini, E. and M. Violante, 2016. Development of an automated test system for ECU software validation: An industrial experience. Proceedings of the 15th Biennial Baltic Conference on Electronics (BEC'16), October 3-5, 2016, IEEE, Tallinn, Estonia, ISBN:978-1-5090-1393-7, pp: 103-106.

Bajeh, A.O., S. Basri, L.T. Jung and M.A. Almomani, 2014. Empirical validation of object-oriented inheritance hierarchy modifiability metrics. Proceedings of the International Conference on Information Technology and Multimedia (ICIMU'14), November 18-20, 2014, IEEE, Putrajaya, Malaysia, ISBN:978-1-4799-5424-7, pp: 189-194.

Balci, O., 1998. Verification, validation and accreditation. Proceedings of the 30th IEEE Conference on Winter Simulation (WSC'98), December 13-16, 1998, IEEE, Washington, USA., pp: 41-44.

Bell, D.G. and G.P. Brat, 2008. Automated software verification and validation: An emerging approach for ground operations. Proceedings of the IEEE Conference on Aerospace, March 1-8, 2008, IEEE, Big Sky, Montana, USA., ISBN:978-1-4244-1487-1, pp: 1-8.

Boehm, B.W., 1976. Guidelines for verifying and validating software requirements and design specifications. Eur. IFIP., 79: 711-719.

Boehm, B.W., 1984. Verifying and validating software requirements and design specifications. IEEE. Software, 1: 75-88.

Briand, L., K.E. Emam and S. Morasca, 1995. Theoretical and empirical validation of software product measures. Intl. Software Eng. Res. Netw., 1: 1-23.

Brooks, F.P., 1986. No silver bullet-essence and accidents of software engineering. Proceedings of the IFIP Congress on Information Processing 86, September 1-5, 1986, Elsevier Science Publishers (BV), Dublin, Republic of Ireland, pp: 1069-1076.

Canfora, G., F. Garcia, M. Piattini, F. Ruiz and C.A. Visaggio, 2005. A family of experiments to validate metrics for software process models. J. Syst. Software, 77: 113-129.

Cruickshank, K.J., J.B. Michael and M.T. Shing, 2009. A validation metrics framework for safety-critical software-intensive systems. Proceedings of the IEEE International Conference on System of Systems Engineering (SoSE'09), May 30-June 3, 2009, IEEE, Albuquerque, New Mexico, USA., ISBN:978-1-4244-4766-4, pp: 1-8.

Csertan, G., G. Huszerl, I. Majzik, Z. Pap and A. Pataricza *et al*., 2002. VIATRA-visual automated transformations for formal verification and validation of UML models. Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02), September 23-27, 2002, IEEE, Edinburgh, England, UK., pp: 267-270.

Dolado, J.J., 2000. A validation of the component-based method for software size estimation. IEEE. Trans. Software Eng., 26: 1006-1021.

Fenz, S. and A. Ekelhart, 2011. Verification, validation and evaluation in information security risk management. IEEE. Secur. Privacy, 9: 58-65.

Goulao, M. and F. Abreu, 2005. Formalizing metrics for COTS. Master Thesis, Department of Informatics, Faculty of Sciences and Technology, New University of Lisbon, Lisbon, Portugal.

Jamil, B., J. Ferzund, A., Batool and S. Ghafoor, 2010. Empirical validation of relational database metrics for effort estimation. Proceedings of the 6th International Conference on Networked Computing (INC'10), May 11-13, 2010, IEEE, Gyeongju, South Korea, ISBN:978-1-4244-6986-4, pp: 1-5.

Johanning, H., J. Lee, C. Hemming and L. Christensen, 2014. Checklist for computer software validation. Pharm. Technol., 38: 56-57.

Kaivola, R., R. Ghughal, N. Narasimhan, A. Telfer and J. Whittemore *et al*., 2009. Replacing Testing with Formal Verification in Intel^{\scriptsize\circledR} CoreTM i7 Processor Execution Engine Validation. In: Computer Aided Verification, Bouajjani, A. and O. Maler (Eds.). Springer, Berlin, Germany, ISBN:978-3-642-02657-7, pp: 414-429.

Khan, A.I., M.M. Alam and U.A. Khan, 2013. Validation of component based software development model using formal B-method. Intl. J. Comput. Appl., 67: 24-35.

Koopman, P., 2015. Verification, validation & certification: Distributed embedded systems. Master Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Li, H., 2016. Validation metrics analysis of community detection algorithms. Proceedings of the 2nd IEEE International Conference on Computer and Communications (ICCC'16), October 14-17, 2016, IEEE, Chengdu, China, ISBN:978-1-4673-9027-9, pp: 2521-2525.

Liu, Y., W. Chen, P. Arendt and H.Z. Huang, 2011. Toward a better understanding of model validation metrics. J. Mech. Des., 133: 1-13.

Mair, C. and M. Shepperd, 2011. Human judgement and software metrics: vision for the future. Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics (WETSoM'11), May 24, 2011, ACM, Waikiki, Honolulu, Hawaii, ISBN:978-1-4503-0593-8, pp: 81-84.

Marculescu, B., 2010. Implementing a software verification and validation management framework in the space industry. Master Thesis, Department of Computer Science and Engineering, University of Gothenburg, Gothenburg, Sweden.

Michael, J.B., M.T. Shing, K.J. Cruickshank and P.J. Redmond, 2010. Hazard analysis and validation metrics framework for system of systems software safety. IEEE. Syst. J., 4: 186-197.

Oberkampf, W.L. and M.F. Barone, 2006. Measures of agreement between computation and experiment: Validation metrics. J. Comput. Phys., 217: 5-36.

Pohl, K. and C. Rupp, 2015. Requirements Engineering Fundamentals. 2nd Edn., Rocky Nook, Santa Barbara, California,.

Ramirez, E.H., R. Brena, D. Magatti and F. Stella, 2010. Probabilistic metrics for soft-clustering and topic model validation. Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'10) Vol. 1, August 31-September 3, 2010, IEEE, Toronto, Ontario, Canada, ISBN:978-1-4244-8482-9, pp: 406-412.

Schneidewind, N.F., 1994. Validating metrics for ensuring space shuttle flight software quality. Comput., 27: 50-57.

Serban, C., A. Vescan and H.F. Pop, 2010. A conceptual framework for component-based system metrics definition. Proceedings of the 9th International Conference on Roedunet (RoEduNet'10), June 24-26, 2010, IEEE, Sibiu, Romania, ISBN:978-1-4244-7335-9, pp: 73-78.

Skroch, O., 2007. Validation of component-based software with a customer centric domain level approach. Proceedings of the 14th Annual IEEE International Conference on Engineering of Computer-Based Systems (ECBS'07), March 26-29, 2007, IEEE, Tucson, Arizona, USA., pp: 459-466.

Sridhar, M. and N.S. Gill, 2015a. Formal conceptual framework for structure of context of component-based system for designing robust software systems and metrics. Intl. J. Comput. Appl., 112: 30-37.

Sridhar, M. and N.S. Gill, 2015b. Imperfection of domain knowledge and its formalization in context of design of robust software systems. J. Software Eng. Appl., 8: 489-498.

Steven, R.R., 2001. Software Verification and Validation for Practitioners and Managers. 2nd Edn., Artech House, Norwood, USA., ISBN:9781580532969, Pages: 387.

Subramani, R., R. Penneru, G. Selvaraj, B. Radhakrishnan and K. Puttaiah, 2014. Coverage metrics for device level validation of SATA and SAS devices: An approach. Proceedings of the 5th International Conference on Intelligent Systems, Modelling and Simulation (ISMS'14), January 27-29, 2014, IEEE, Langkawi, Malaysia, ISBN:978-1-4799-3858-2, pp: 163-168.

Szabo, C. and Y.M. Teo, 2012. An integrated approach for the validation of emergence in component-based simulation models. Proceedings of the Conference on Winter Simulation (WSC'12), December 9-12, 2012, ACM, Berlin, Germany, pp: 242-242.

Tomar, P. and N.S. Gill, 2010. Verification and validation of components with new x component-based model. Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE'10) Vol. 2, October 3-5, 2010, IEEE, San Juan, Puerto Rico, USA., ISBN:978-1-4244-8667-0, pp: V362-V365.

Tripathi, P., M. Kumar and N. Shrivastava, 2008. Theoretical validation of quality metrics. Proceedings of the 2008 International Conference on Software Engineering Research and Practice (SERP'08), July 14-17, 2008, Monte Carlo Resort, Las Vegas, Nevada, USA., pp: 1-7.

Usai, F., K. O'Neil and A.J. Newman, 2017. Design and empirical validation of effectiveness of LANGA an online game-based platform for second language learning. IEEE. Trans. Learn. Technol., 99: 1-1.

Veerappa, V. and R. Harrison, 2013. An empirical validation of coupling metrics using automated refactoring. Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, October 10-11, 2013, IEEE, Baltimore, Maryland, USA., ISBN:978-0-7695-5056-5, pp: 271-274.

Venkitachalam, H., J. Richenhagen, A. Schlosser and T. Tasky, 2015. Metrics for verification and validation of architecture in powertrain software development. Proceedings of the 1st International Workshop on Automotive Software Architecture (WASA'15), May 4-8, 2015, IEEE, Montreal, Canada, ISBN:978-1-4503-3444-0, pp: 27-33.

Woodcock, J., P.G. Larsen, J. Bicarregui and J. Fitzgerald, 2009. Formal methods: Practice and experience. ACM. Comput. Surv., 41: 1-36.

Xargay, E., V. Dobrokhodov, I. Kitsios, I. Kaminer and K.D. Jones *et al*., 2009. Flight validation of a metrics driven L1 adaptive control in the presence of general unmodeled dynamics. Proceedings of the IEEE International Conference on Control and Automation (ICCA'09), December 9-11, 2009, IEEE, Christchurch, New Zealand, ISBN:978-1-4244-4706-0, pp: 2243-2248.

Yan, S., Y. Zhao and P. Chen, 2015. Automated test platform for FPGA software validation. Proceedings of the International Conference on Semiconductor Technology (CSTIC'15), March 15-16, 2015, IEEE, Shanghai, China, ISBN:978-1-4799-7242-5, pp: 1-3.

Zhixiong, J., Q. Leqiu and P. Xin, 2007. A formal framework for description of semantic web services. Proceedings of the 7th IEEE International Conference on Computer and Information Technology (CIT'07), October 16-19, 2007, IEEE, Aizu-Wakamatsu, Fukushima, Japan, ISBN:978-0-7695-2983-7, pp: 1065-1070.