www.ijecs.in

International Journal Of Engineering And Computer Science ISSN:2319-7242

Volume 4 Issue 4 April 2015, Page No. 11246-11251

Maintainability assurance in Software Product Lines An Activity- Based Model

Manoj Nainwal[1], *Anurag Awasthi*[2]

[1]Singhaniya University, School of Computer Science,

PacheriBari(Junjhunu), Rajasthan, India

Manoj.nainwal13@email.com

[2]School of Computer Science and Engineering, Chung-Ang University,

221, Heukseok-dong, Dongjak-gu, Seoul 156-756, Korea

author2@email.com

***Abstract*—** Management of maintainability of software in Software Product Lines is still a problematic area. As its very important quality attribute, there has to be a comprehensive basis for assessing and improving it. Several quality models have been proposed to quantify the maintainability. Nevertheless, existing approaches are not activity based. We have proposed a set of activities and set of facts to calculate maintainability in SPL (Software Product Lines) environment. Facts would have a cascaded effect on activities which in turn would have an effect on the sub-factors of maintainability. We have conceptualized the set of activities and facts in the form of an activity based model

***Keywords*—**Activity Based Model, Binding Time, Granularity, Multiplicity, Software Product Lines (SPL), Variability

WITH reference to the latest software quality model proposed by ISO (International Standard Organization) i.e. ISO/IEC 9126 model, maintainability is one of the six characteristics. Other characteristics are Functionality, Efficiency, Portability, Reliability and Usability. All these characteristics are having their own sub characteristics [13].Since the announcements of ISO/IEC 9126 model; many researchers have added various dimensions to this model. A new model has been designed for evaluating the software quality. [3][15] gives the reference of activity- based quality model. In addition to this model Bayesian network approach has been used as an effective quantification approach to quantify various attributes of ISO/IEC 9126 model. A lot of research has been done to ensure better quantification.

Software Product Lines are an attractive solution for software development because they are economical and adaptable to change. Clements and Northrop [17] defines SPL as "a set of software intensive systems that share a common, managed set of features and that are developed from a common set of core assets in a prescribed way." The selection of a feature in a product can occur at different times, as required by many development issues [18]. In this paper I have conceptualized the maintainability of software product line. In view of activity based quality model maintainability will have implementation effects of four major dimensions i.e. variability, granularity, multiplicity and binding time. These four major dimensions are defined in software product lines development. I have considered them as facts in activity based model and conceptualized their effects on activities to ensure maintenance or maintainability of SPL. This paper is organized as follows. Section 1 is discussing the ISO/IEC 9126 model. It also discusses the maintainability and its sub characteristics with the extended factors of each sub factors which show the activities to ensure the relevance of sub factors to measure the maintainability. Section 2 shows the activity- facts based model and discusses the concept to have a better understanding. Section 3 discusses the theoretical and mathematical basis to apply the activity-facts based model to quantify the maintainability of SPL. Section 4 discusses the future work, limitations and conclusions.

Efforts based quality measurement models are very common. In these process models roles, activities and artifacts play a very important role. Quality model such as ISO 9126 is based on six parameters like functionality, reliability, usability, efficiency, maintainability and portability. Table 1 shows the characteristics and associated sub- characteristics mentioned in ISO/IEC 9126 model. I have considered this model as a base model for this paper and for this study. In this model maintainability has four major sub characteristics named analyzability, changeability, stability and testability.

TABLE 1

Characteristic	Sub Characteristics
Functionality	Suitability
	Accurateness
	Interoperability
	Compliance
	Security
Reliability	Maturity
	Fault tolerance
	Recoverability
Usability	Understandability
	Learnability
	Operability
	Attractiveness
Efficiency	Time Behaviour
	Resource utilization
Maintainability	Analyzability
	Changeability
	Stability
	Testability
Portability	Adaptability
	Installability
	Conformance
	Replaceability

Table 2 describes the maintainability and its sub-factors expressed in ISO/IEC 9126. It also shows the proposed set of activities related to each sub- factors of maintainability.

TABLE 2

REFERENCE TABLE TO EXPRESS MAINTAINABILITY AND ITS SUB FACTORS

Quality **Factor**	**Sub- Factors**	**Proposed set of activities**
Maintainability	Analyzability	Code structure
		Code readability
		Code complexity
	Changeability	Document Readability
		Implementation efforts
		Documenting efforts
	Stability	New requirements
		Enhancements
		Optimization
	Testability	Observability of code
		Isolation

The contribution of these six parameters has some variations with regard to the application domain and the software company's internal methods to maintain and develop software. We have proposed a set of activities related to each sub-factor. We have also proposed that these set of activities affect the sub-factors and in turn the maintainability gets affected. In order to quantify the values of sub-factors we have assumed that all these sub-factors and set of activities are variables. In section 3 we will present a mathematical model to quantify all of them.

While measuring quality of a software product line four major dimensions need to be taken care. These four dimensions are:

1. Variability
2. Granularity
3. Multiplicity
4. Binding-time

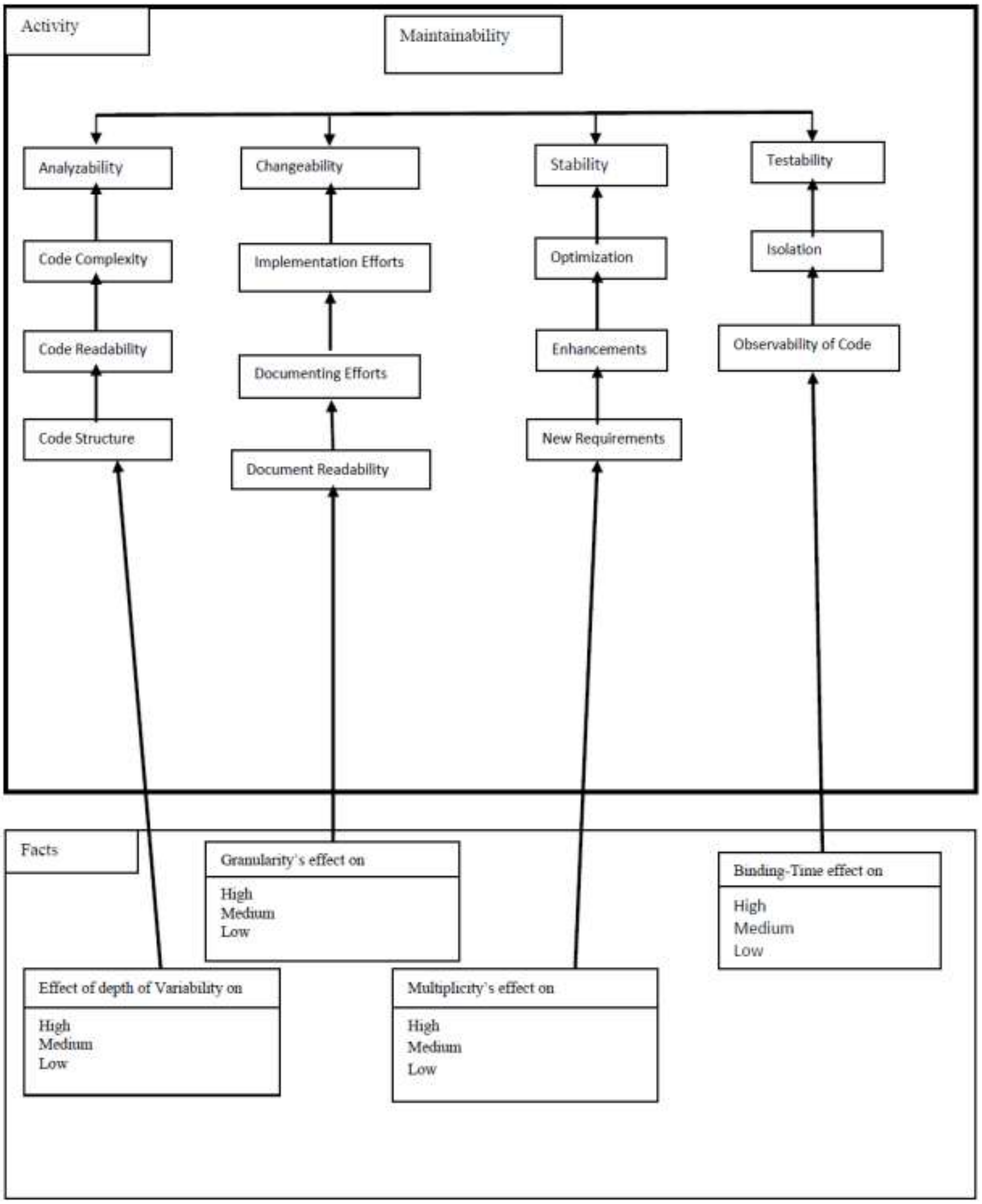
These four dimensions (facts) affect software quality differently at different phases of software development. At design level analyzability depends on code structure. This code structure dictates the code readability and in turn code readability defines the code complexity of the software. Therefore in this paper we have taken these three as the parameters to evaluate the analyzability. At the same time these three factors related to code can be affected by the variability inherently a necessary feature for software product line engineering. In practice, variability can be observed in code, in functions, in control flows, which we have defined in next paragraph. Now a day's feature rich applications are in demand [12]. This demand is catered by the software engineers by adopting the reusable components of a suite of software packages. Future is the cloud based computing when a software engineer will be able to find a component on cloud and will be able to integrate it with its ongoing software.

Another very interesting advancement in software development field is the evolution of software product lines. Now a company produces a feature oriented software packages and its extensions over the period of time. As features grow the quality of software product becomes a very important aspect. In order to maintain the overall quality, SPL quality metrics need to be developed. For SPL, the quality metrics will be different in the sense that already existing feature and its quality may be affected by the addition of or removal of another feature of package.

In this paper we are focusing on the aspects which may affect the quality of software package in feature oriented SPL development process. We will investigate granularity, variability, multiplicity and binding time dependency. It has been proposed to use activity-based quality models (ABQM) in order to assess the quality **[9].**

Here we have defined four sub-factors to assess maintainability. These four sub-factors are further having activities related to each sub-factors. We are proposing the four above mentioned major dimensions as the facts because these dimensions affect different major activities. We are proposing that that these four attributes affect sub-factors of maintainability as mentioned below:

Variability → Analyzability i.e. variability has a direct effect on analyzability.



ACTIVITY –FACTS BASED MODEL

Granularity → Changeability i.e. granularity has a direct effect on changeability.

Multiplicity → Stability i.e. multiplicity has a direct effect on stability.

Binding Time → Testability i.e. binding time effects testability.

Four dimensional attribute of quality of software are:

1. *Variability*- David M. Weiss and Chi Lai define

variability in product family development as “An assumption about how members of a family may differ from each other".

Variability in data can be observed as a particular data structure may vary from one product to another. Variability in control flow is defined as variation of interaction pattern from one product to another. Variability in function shows that some functions may exist in some products and not in others. Variability in product quality goals may be understood as the variations of goals like security, performance or modifiability from one product to another. Variability in product environment means that the product domain may impose specific requirements. Variability in technology concerns the platform (OS, hardware, dependency on middleware, user interface, and run-time system for programming language) which may vary in a similar way to function but with the technical point of view [22][23].

By and large variability affects different aspects of quality of software in SPL. In this paper we have proposed that uncertainty in variability can affect the analyzability of software, which in turn affects the maintainability of software. In this paper we are considering the technical variability and product level variability [22] [10].

2. *Granularity*- In Software Product Line development main focus of the developer is on feature addition. Fine grained coding practices make this easy [11] [22] [23]. [27][28] explored the limitations in implementing fine grained extended features. We have suggested that fine grained extensions need extra documenting efforts. But it makes document more readable and in turn reduces the implementation efforts in due course of development process. In compositional approach to implement SPL statement extensions [28] [27], expression extensions, and signature changes show obvious limitations. In annotative approach to implement SPL shows obfuscated source code and problem in annotating arbitrary code fragment without considering its relevance [29] [30]. Hence we are proposing that high granularity will provide better feature management. Thus, granularity may affect the overall maintainability of SPL.
3. *Multiplicity*- This is also known as the attribute of a variation point. Multiplicity of a variation point indicates the minimum number of elements of the associated variant set. Variant set selection depends on its multiplicity which can vary from 0 to 1.
4. *Binding Time*- It refers to the time at which the decisions for a variation point are set. [19] In other words it is the time at which features selection occurs. There are many different binding times available to a software designer e.g., compile time, link time, and run time. A software development technique that is used to implement a variation point is called variability mechanism [20]. In a continuously evolving product line, where binding times may change based on existing, evolving, or expanding domain requirements, this process is error-prone and the code modifications are tedious to track [21].

This section discusses the theoretical basis for activity facts based model. We have assumed that at activity level four major activities are there to ensure maintainability of software. Under each major activity some sub activities are defined. For example major activity “analyzability” can be measured by 3 sub activities like “code complexity”, “code readability”, and “code structure”. Here “code structure” is the lowest sub activity. We have shown the effect with “+” and “-“signs. “+” sign signifies the increased impact of the fact on activities related to sub-factors. “-“sign signifies the decreased impact.

We include 4 impacts on each of the low level activities. These facts together with their impacts are:

1. [VARIABILITY|DEPTH] $\xrightarrow{+}$ [CODE COMPLEXITY]

The depth of variability will affect the level of code complexity. As depth of variability will increase it will increase code complexity or we can say that if code is complex its affect will be directly reflected on the variability. To implement high variability, which is a basic feature of any software product line, code must be simple. To ensure high variability you have to make coding simple as it can be achieved by low coupling.

$$\begin{tikzcd}
\text{[CODE COMPLEXITY]} \arrow[d, "-"] \\
\text{[CODE READABILITY]} \arrow[d, "-"] \\
\text{[CODE STRUCTURE]}
\end{tikzcd}$$

In the above diagram it is shown that code structure will become complex if code readability is low. Similarly if code readability is low code becomes very complex. This decreases the variability.

2. [GRANULARITY | EFFORTS] $\xrightarrow{\_}$ [DOCUMENTING EFFORTS]

The granularity directly affects the documenting efforts as more granular the software documentation becomes large.

$$\begin{tikzcd}
\text{[DOCUMENTING EFFORTS]} \arrow[d, "+"] \\
\text{[DOCUMENTING READABILITY]} \arrow[d] \\
\text{[IMPLEMENTATION EFFORTS]}
\end{tikzcd}$$

3. [MULTIPLICITY | VARIANCE] $\xrightarrow{+}$ [NEW REQUIREMENTS]

New requirements need to be added to software to cope up with the changing scenario, makes software having positive multiplicity.

$$\begin{tikzcd}
\text{[NEW REQUIREMENTS]} \arrow[d] \\
\text{[ENHANCEMENTS]} \arrow[d] \\
\text{[ERROR CORRECTION]} \arrow[d] \\
\text{[OPTIMIZATION]}
\end{tikzcd}$$

4. [BINDING-TIME | TIME INTERVAL] $\xrightarrow{-}$ [OBSERVABILITY OF CODE]

This means binding time interval inversely affects the observability of code.

[9] have discussed the activity based model for maintenance of the software. This work has been the basis of my work in context with SPL. [12] has discussed the variability management as a challenge in Software Product Lines. [7] [8] have discussed the multiplicity and its management in SPL. [24] has through a light upon the term SPL and its definitions. [21] [22] have discussed the principles and practices in SPL development environment.

In this paper a new approach to quantify the measurement of maintainability in SPL is proposed. We have proposed the activity based concept to analyze the effects of variability, granularity, multiplicity and binding time on SPL maintainability. We are currently working on the model to couple it with Bayesian belief network to make it more empirical. By incorporating fuzzy logic in this model, we can propose an industry ready model to assure maintainability in SPL.

REFERENCES

[1]. Efficient estimation of effort using machine-learning technique for software cost S. Malathi,S. Sridhar, Indian Journal of Science and Technology Vol. 5 No. 8 (August 2012) ISSN: 0974-6846

[2]. enhancing software reliability of a complex software system architecture using artificial neural-networks ensemble, parmod kumar kapur, v. s. sarma yadavalli International Journal of Reliability, Quality and Safety Engineering Vol. 18, No. 3 (2011) 271–284_c World Scientific Publishing Company

[3]. A Bayesian Network Approach to Assess and Predict Software Quality Using Activity-Based Quality Models,Stefan Wagner, Fakultät für Informatik ,Technische Universität München, Garching b. München, Germany

[4]. Requirement Risk Level Forecast Using Bayesian Networks Classifiers Isabel Mar´Ia Del ´Aguila International Journal of Software Engineering and Knowledge Engineering Vol. 21, No. 2 (2011) 167–190_c ,World Scientific Publishing Company

[5]. A Study of Variability Spaces in Open Source Software ,Sarah Nadi,David R. Cheriton School of Computer Science, University of Waterloo, ON, Canada, 978-1-4673-3076-3/13, 2013 IEEE

[6]. Quantifying the Stability of Software Systems via Simulation in Dependency Networks Weifeng Pan, Member, ACM, World Academy of Science, Engineering and Technology 60 2011

[7]. Multiplicity Computing:A Vision of Software Engineering for Next-Generation Computing Platform Applications
Cristian Cadar, Peter Pietzuch, and Alexander L. Wolf
Department of Computing, Imperial College London, London, UK

[8]. David M. Weiss and Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process. AddisonWesley, 1999.

[9]. F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert,and J.-F. Girard. An activity-based quality model for maintainability. In Proc. 23rd International Conference on Software Maintenance (ICSM '07). IEEE Computer Society Press, 2007.

[10]. S. Jarzabek and L. Shubiao. Eliminating Redundancies with a "Composition with Adaptation" Meta-Programming Technique. In ESEC/FSE. 2003..

[11]. G. Murphy et al. Separating Features in Source Code: an Exploratory Study. In ICSE. 2001.

[12]. M. Rosenmüller, M. Kuhlemann, N. Siegmund, and H. Schirmeier. Avoiding Variability of Method Signatures in Software Product Lines: A Case Study. In GPCE Workshop on Aspect-Oriented Product Line Engineering, 2007.

[13]. ISO/IEC 9126-1:2001, Software Engineering-Product Quality—Part 1:Quality Model, Int'l Organization for Standardization, 2001, Available at www.iso.org.

[14]. M. Neil, B. Littlewood, and N. Fenton. Applying bayesian belief networks to systems dependability assessment. Safety Critical Systems Club Newsletter, 8(3), 1999.

[15]. N. E. Fenton, M. Neil, and J. G. Caballero. Using ranked nodes to model qualitative judgments in Bayesian networks. IEEE Transactions on Knowledge and Data Engineering, 19(10):1420-1432, 2007.

[16]. F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert,and J.-F. Girard. An activity-based quality model for maintainability. In Proc. 23rd International Conference on Software Maintenance (ICSM '07). IEEE Computer Society Press, 2007.

[17]. P. Clements and L. Northrop. Software Product Lines: Practices and Patterns. Addison Wesley, 2001.

[18]. M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. Software—Practice & Experience, 35(8):705–754, 2005.

[19]. E. Dolstra, G. Florijn, and E. Visser. Timeline variability: The variability of binding time of variation points. In Proc. of the 2003 Workshop on Software Variability Management (SVM), pages 119–122, Gronigen, The Netherlands, Feb. 2003.

[20]. I. Jacobson, M. Griss, and P. Jonsson. Software Reuse: Architecture,Process and, Organization for Business Success. Addison Wesley,1997.

[21]. R. van Ommering. Building product populations with software components. In Proc. of the 24th International Conf. on Software Engineering (ICSE), pages 255–265, Orlando, FL, May 2002.

[23]. Analysis of a software product line architecture: an experience report, Robyn R. Lutz , Gerald C. Gannod, The Journal of Systems and Software 66 (2003) 253–267, Elsevier Science Inc.

[24]. Gannod, G.C., Lutz, R.R., Cantu, M., 2001. Embedded software for a space interferometry system: Automated analysis of a software product line architecture (invited). In: Proceedings of the International Conference on Performance Computing and Communications, pp. 145–150.

[25]. ISO, International Organization for Standardization, "ISO 9000-2:1997, Quality Management and Quality Assurance Standards —Part 2: Generic guidelines for the application of ISO 9001, ISO 9002 and ISO 9003", 1997.

[26]. J. Liu, D. Batory, and C. Lengauer. Feature Oriented Refactoring of Legacy Applications. In ICSE, 2006.

[27]. C. Kästner, S. Apel, and D. Batory. A Case Study Implementing Features Using AspectJ. In SPLC, 2007.

[28]. G. Murphy et al. Separating Features in Source Code: an Exploratory Study. In ICSE. 2001.

[29]. H. Spencer and G. Collyer. #ifdef Considered Harmful or Portability Experience With C News. In USENIX Conf., 1992.

[30]. Baxter and M. Mehlich. Preprocessor Conditional Removal by Simple Partial Evaluation. In Proc. Working Conference on Reverse Engineering. 2001.

[31]. ISO, International Organization for Standardization, "ISO 9000-3:1998-Quality Management and Quality Assurance Standards – Part 3: Guidelines for the application of ISO 9001_1994 to the development, supply, installation and maintenance of computer software (ISO 9000-3:1997)", 1998.

[32]. ISO, International Organization for Standardization, "ISO 9004:2000,Quality Management Systems-Guidelines for performance improvements", 2000.

Manoj Nainwal is an MCA from IMS, Dehradun and has more than 10 years of industry and teaching experience. He is pursuing Ph. D. from Singhania University, Rajasthan on the topic "Framework Development to ensure security in Database Management Systems-A Refactoring

Approach". He has taught Software Engineering, RDMS, Object Oriented Analysis and Design and Network Security to Graduate and Post Graduate courses. He can be contacted at manoj.nainwal13@gmail.com.

Anuraag Awasthi is former Dean (Engineering & Technology), JV Women's University, Jaipur, former Director (MBA), IIMT, Dehradun. He has more than 23 years experience, including 21 in industry, with his last designation as AVP(HR) with Bharti Airtel group. He has led functions like HR, Software Development, Quality, Testing, Information Security, Customer Support and IT. He has a wide hands-on exposure in all sectors – Govt., and Private, India and Abroad. He has a passion for Excellence in all spheres of life. He is a CSQA, Six Sigma Black Belt, MCA, M.Sc. (TQM), PGDHRM and PhD in Computers with specialization in Software Quality. He can be reached at Anuraag_awasthi@hotmail.com.