International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 09 | Sep-2016 www.irjet.net p-ISSN: 2395-0072

Modifiability Analysis at Architecture Level through Scenario Elicitation (MAASE)

P. Ashok Reddy[1], Dr. K. Rajasekhara Rao[2], Dr. M. Babu Reddy[3*]

[1] Sr. Asst. Professor of Comp. Applications LBRCE-Mylavaram, Krishna Dt., AP, India
[2]Director, Usha Rama College Of Engineering & Technology, Vijayawada, AP, India
[3*]Asst. Professor of Computer Science Krishna University, Machilipatnam, AP, India

---

Abstract:-It is evident from the studies that majority of the total lifecycle cost is consumed by system evolution. Organizations are now concentrating on reducing the modifiability during the system development itself. In the process of achieving this, software architecture plays vital role at each level of modifiability analysis. Researchers have proposed many methods that helps to assess software architecture with respect to various quality attributes. Research has also been carried out to identify the rigidness at the software architecture level. As user scenarios and experience of the user are basic characteristics of architecture understanding and analysis, the proposed research paper emphasized the elicitation and evaluation of scenarios while analyzing the modifiability at architecture level. A generalized framework of Online Banking Application (OBA) has been taken with most frequently used scenarios as case study to illustrate the architecture centric modifiability model.

Key words: *Modifiability Analysis, Scenario, Software Architecture*

1. Introduction

It is proved from way of the studies that around 70% [Eckland et al..][lientz and swanson] of the total life cycle cost is spent on system evolution. Organizations are trying to reduce modifiability during the systems development, by looking into it during the systems development itself. To achieve this, software architecture plays vital role and few methods were existed for each level of modifiability analysis. To predict the quality attributed, software architecture need to be analyzed. Kazman etal proposed Software architecture analysis method (SAAM) which is aided at assessing software architecture with respect to different quality attributes. The key idea of SAAM is based on the user scenarios and the experience of user is taken into priority. Bengtsson and Bosch have tried to forecast the required maintenance effort based on the architecture of the software and Lassing et al.. defined a method to identify the rigidness at the software architecture level. Different methods differ based on the assumptions which will be influenced by the goals and by combining various methods these assumptions led to unified architecture level modifiability analysis method.

The key steps of architecture level modifiability analysis include the description of relevant architecture as per the goal scenario elicitation, evaluation and interpretation. Depending on the goal, focus can be diverted to the some of the key steps [P. Bengtsson and J. Bosch]. The method has been applied fruitfully to the software applications like online banking portals, social networking sites and recently introduced web based counseling of Andhra Pradesh State Government.

2. Structure of MAASE

The structure of MAASE consists of the following essential five steps:

a) Goal setting: determine the aim of the analysis
b) Software Architecture Description: give a description of the relevant parts of the software architecture
c) Scenario Elicitation: find the set of relevant scenarios
d) Evaluate scenarios: determine the effect of the set of scenarios
e) Results Interpretation: draw conclusions from the analysis results

When performing an analysis, the separation between the tasks is not very strict and it is often necessary to iterate over various steps [PerOlof Bengtsson et al..]. However, the paper presents the steps as if they are performed in strict sequence.

2.1 Goal Setting

The first activity in MAASE is concerned with determining the goal of the analysis. In any analysis that is connected to modifiability at architecture-level, the following goals can be pursued.

- Prediction of Maintenance cost: Estimate the effort that is required to accommodate future changes
- Risk assessment: Recognize the types of changes for which the software architecture is not flexible
- Selection of optimum architecture for the chosen software.

2.2 Architecture description

In the second step of MAASE, the information about the software architecture is collected. Generally speaking, modifiability analysis requires architectural information that allows the analyst to evaluate the scenarios which is concerned with the analysis of the role and influence of the scenarios and expressing this. The key objective of the impact analysis at Architecture-level is to recognize the architectural elements exaggerated by a change scenario. Based on the goal of the modifiability analysis, a measurement scale can be used to express the scenarios.

While analyzing the impact at architecture-level [PerOlof Bengtsson et al..] the views that are used should present information about:

• The decomposition of the system in components
• The relationships among various components
• The relationship of the system with its environment

The individual components of the architecture can be viewed as a functional breakdown of the system, i.e. various functions of the system will be shared among various components or functional units. Relationships among various components and between the system and its environment come in different forms and are often defined implicitly. It is vital to impact analysis as they conclude whether modifications to a functional element will reason in ripple effects or not. The probability of identifying dependencies among components at the software architecture level is limited. Some of them are introduced during detailed design and implementation. These relationships may lead to sudden ripple effects when the components are customized. As a result, it is not always possible to find out the full impact of a scenario and this influences the overall accuracy of analysis.

During the initial stages, the available information can be used for analysis. If the information is not enough to look into the consequence of the scenarios found, the absent information can be filled with the help of a designer or architecture.

2.3 Scenario Elicitation

Scenario elicitation is the process of finding and selecting the scenarios that are connected to changes to be used in the evaluation step. Eliciting change scenarios involves activities such as identifying stakeholders to interview, documenting the scenarios that result from those interviews, etc. Although the elicitation shares some common aspects with other fields, e.g. in requirements engineering, scenarios are elicited from stakeholders and documented for later use as well (Sutcliffe et al.,), and there are some issues of specific concern.

The first issue is that the number of possible changes to a system is almost infinite. In order to make scenario-based software architecture analysis feasible, a combination of two techniques: (1) 'equivalence classes' and (2) 'classification of change categories' can be used. Partitioning the space of scenarios into equivalence classes enable us to consider one scenario as a representative of a group of scenarios, thus limiting the number of scenarios that have to be considered.

However, not all equivalence classes are just as relevant for each analysis. Deciding on important change scenarios requires a selection criterion. The other technique, classification of change categories, is used to focus attention on the scenarios that satisfy this selection criterion.

For selecting the scenarios, top-down and bottom-up approaches can be used.

To-down approach will be guided by some predefined set of change categories while searching for change scenarios. The source of this classification is from the chosen application domain, knowledge about complex scenarios, or some other external sources of knowledge. This kind of classification helps in bringing forward the relevant scenarios from the stakeholders. But, while using a bottom-up approach, a predefined classification scheme will not be available, and the stakeholders being interviewed will come up with a relevant set of scenarios. The analyst then categorizes the scenarios and it will be reviewed by the stakeholders. The bottom up strategy will result in an explicitly defined and colonized set of change categories.

In day to day application, the architects often iterate between the approaches such that the extracted set of change scenarios are used to construct or improve the classification scheme which in turn used to guide the process of searching for additional

scenarios. In addition to the scenario selection criterion, a stopping criterion is also required. Sufficiency can be achieved in number of change scenarios when: (1) all the change categories are explicitly considered and
(2) newly identified change scenarios do not affect the classification structure.

- If the goal is to estimate maintenance effort, scenarios that have higher probability of occurring changes during the operational life of the system need to be considered.
- If risk assessment is the key goal, scenarios that expose those risks need to be considered.
- If the goal is to study the differences between various architectures, emphasis is required on scenarios that bring to light the differences among those architectures.

But, there is no need to expose the boundary conditions of the considered systems those involving different owners. MAASE uses two mechanisms to organize the set of scenarios as well as the elicitation process: utility trees and facilitated judgment. Utility trees provide a top-down mechanism guided by the top down decomposition of applicable quality attributes in the evaluation. The proposed top-down mechanism only concerned with modifiability, and is structured in a different way. Facilitated judgment is a bottom-up mechanism like the one that will be employed.

2.4 Evaluation of Change scenarios

While evaluating the effect of change scenarios on the software architecture, the analyst will work together with software architects and the designers. As a team they determine the impact of the change scenarios and the results will be expressed in a way that is suitable for the objective of the chosen analysis. The primary steps of Impact analysis are:

1. Identify the affected components or data and functional elements
2. Find out the effect on the components
3. Identify the ripple effects

The first step is to determine the components that need to be modified to implement the change scenario. The second step is concerned with the identification of the functionality of the components that are affected by the changes. Changes may also cross the system boundaries; changes in the environment may have influence on the system or vice versa. Hence, the systems in the environment and their interfaces should also be considered. The third step is to determine the ripple effects of the identified modifications. The occurrence of ripple effects can be viewed as a recursive phenomenon as each ripple may have additional ripple effects. Since limited or incomplete information is available at the architecture level, assumptions need to be made out about the occurrence of ripple effects[PerOlof Bengtsson and et al..].

Regarding the existence of ripple effect, the professionals normally will not look into too optimistic or too pessimistic. In practice, the architects and designers play vital role in determining whether revisions to a component have ripple effects on other components.

2.5 Interpretation

After the finalization of change scenario evaluation, results need to be interpreted to draw conclusions with reference to the software architecture. The goal of the analysis and the system requirements will influence the interpretation of the results. More realistic design can be ensured through the usage of scenarios for maintenance effort prediction as scenarios have emerged as a considerable means at the earlier stages of system life cycle. This helps in reducing overall effort required for maintenance significantly.

Giving focus to modifiability

- differentiate numerous analysis goals
- Make hidden assumptions open
- present well-documented techniques for performing the required steps of analysis

As the analysis of each quality attribute has its specific issues and challenges, the proposed paper mainly focuses on modifiability. In order to get a full evaluation of software architecture, MAASE should be used along with additional quality attributes.

Some of the researchers have come up with similar methods of scenario based architecture analysis, but were built on different assumptions. These assumptions were mostly prejudiced by the domains and the objectives pursued in the analyses. The differences in the goal of the analysis led to different techniques to be used in the various steps of an analysis, i.e. differences in required information, different means of scenario elicitation (including the notion of a 'more appropriate scenario') and various methods of scenario evaluation.

3. Maintenance Prediction- Case Study-1

A case study has been conducted to illustrate the proposed modifiability Architecture model on online banking application (OBA). A generalized operational frame work of online banking portal has been taken with the most frequently used scenarios.

a) Fixing the goal

The goal of analysis of online banking application is to get near estimates or qualification prediction of the costs of modifying the system for future changes. For this, the possible scenarios that may occur in the future need to be identified, amount of changes required and a model to derive the prediction need to be worked out. The following sub sections describe the working of proposed model.

b) Architecture Description

For maintenance prediction, information is required for estimating the amount of modifications required to realize the scenario. To identify the specific parts of architecture where changes are required, logical view(P.B.Kurchen et al,[12]) or conceptual view(Hofmeister et al [13]) may be used. After identification of modules to be modified, ripple effort was determined .The same metrics that are in use for estimating time and effort in the planning were used here also. In addition to the architecture information, Size estimates of the components were also taken and same unit of size as it used for the productivity measures in the project.

In Online Banking Application Project, the architecture information was collected from the employee's of three different banks who were working in software maintenance divisions of respective banks and due to the security limitations only the essentials of architecture is collected and basing on the conservations with these employees, it is understood that the architecture descriptions contained the following views:

- ✓ Systems relation with the environment
- ✓ Behavior of the key functions in the system through sequence diagram
- ✓ Layer wise organization of the system especially focusing on conceptual view.
- ✓ View of components

The required rough size estimates of individual components were obtained from the architecture of similar application domains from various software development firms. The architects used the previous projects data and their experience as reference which estimating the size.

c) Elicitation of change scenarios

To elicit the scenarios which may undergo changes, stake holders were indentified to interview for identification of scenarios. It is important to select the persons who have wide range of experiences on the systems future changes as possible. The compulsory stakeholder areas that need to be considered are: Architecture design, customer training, product management and maintenance. As the chosen application deals with the large number of public, individual knowledge about customer needs and interests, marketing and sales areas need to be considered.

Bottom-up elicitation technology is preferred as the stakeholders have the better knowledge, about likely scenarios. In the process of scenario elicitation, scenarios were classified and redundant scenarios were removed. The stakeholders are asked to prioritize the categories and identify the scenarios which can be eliminated from further consideration. And the stakeholders also asked to identify the important scenarios that were missed. After few iterations and by thorough brainstorming by the stakeholders group, the following change scenario categories were indentified.

- ➢ User interface updation at regular intervals
- ➢ Operational quality requirements
- ➢ Continuous enhancement of security features
- ➢ Addition of new functionalities
- ➢ Interoperability provisions

For better estimates of predictions, probability of scenario occurrence must also be estimated which is used to determine the influence of scenario on the end results.

By interviewing the appropriate stakeholders, the tentative frequency of each scenario is identified and the suitable weights are assigned to scenarios. After assigning scenarios, scenario profile is created by normalizing scenario weights.

$N(S_i) = S_{iw} / \sum S_{iw \text{ for all } j=1 \text{ to } n}$

S_{iw}= weight of i^{th} Scenario

$N(S_{iw})$ = Normalized scenario weight

Because of this normalization, the weights of scenarios will fall in between 0 and 1 and the sum of all the scenarios weights will equal to 1. The following table illustrate the calculation of average effort per scenario.

Change Scenario	Weight	Impact
UT Change	0.42	5% of 3.5 KLOC
Security Enhancement	0.33	7% of 3.5 KLOC
New Functionalities	0.15	10% of 3.5 KLOC
Effects of identified charges	0.075 (Cumulative)	5% of existing KLOC
Ripple Effect	0.025	0.2 % of total no. of functions

The size of the modification to existing components can be derived from an estimate of the ratio of the change to modified components. With an assumption that writing new components gives better productivity than that of modifications made to existing modules, separate analysis and representation is followed for each category of modification.

4. Interpretation & Conclusion

The primary requirement of prediction of maintenance effort requires a model which is driven by cost drivers. In the proposed study, volume of product/component change is considered as key cost driven. Total effort needed for a change in scenario profile is calculated as

$TE = (\sum_{CCi}(Size_i \text{ X } Weight_i))\ P_{cc} + (\sum_{NCi(}Size_i \text{ X } Weight_i))\ P_{NC}$

Where TE= Total Effort

CC_i= Changed code in component

NC_i= New Code in component

P_{cc} = Productivity connected to change code

P_{NC}= Productivity connected to new code

The TE value estimated for individual components can be summed up for all changes and the total effort required for the entire project can be estimated.

Here, in this section of paper, a prediction model was presented based on the estimated change volume and productivity ratios. The techniques are mostly based on the experience of various domain experts. Though the results are facilitating for good estimates of effort, it lack a frame of reference. The alarming issue is how best the results are in comparison to counterpart alternatives. For this, a frame of best and worst case references are also drawn from the same data.

References:

[1] John, B. E., & Marks, S. J. (1997). Tracking the effectiveness of usability evaluation methods.

[2] M.S.Dresselhaus, G.Dresselhaus and P.C.Eklund, science of fullerness and carbon nanotubes, chapter13,Academic Press (1996).

[3] Lientz and Swanson, Software Maintenance.

[4] R. Kazman, G. Abowd, L. Bass, P. Clements,Scenario-based analysis of software architecture**,** IEEE Software, 13 (6) (1996), pp. 47–56.

[5] Bengtsson and Bosch An experiment on creating scenario profiles for software change.

[6] PerOlof Bengtsson a, Nico Lassing b , Jan Bosch c , Hans van Vliet,Architecture-level modifiability analysis (ALMA) at elsewier Inc.All , The Journal of Systems and Software 69 (2004) 129–147.

[7] P. Bengtsson, J. Bosch,Architecture level prediction of software maintenance**-**
Proceedings of 3rd EuroMicro Conference on Maintenance and Reengineering (CSMR'99)IEEE CS Press, Los Alamitos, CA (1999), pp. 139–147.

[8] Sutcliffe, A.G., Maiden, N.A.M., Minocha, S., Manuel, D., 1998. Supporting scenario-based requirements engineering. IEEE Transactions on Software Engineering 24 (12), 1072–1088.

[9] Bohner, S.A., 1991. Software change impact analysis for design evolution. In: Proceedings of 8th International Conference on Maintenance and Re-engineering. IEEE CS Press, Los Alamitos CA, pp. 292–301.

[10] Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., Chen, C., 1994. Change impact in object oriented software maintenance. In: Proceedings of the International Conference on Software Maintenance (ICSM'94). IEEE CS Press, Los Alamitos, CA, pp. 202–211

[11] Nico Lassing,Architecture-Level Modifiability Analysis

[12] P.B.Kurchen, " The 4+1 view Model of Architecture",IEEE Software, pp 42-50,November 1995.

[13] Hofmeister E, Ellis BA, Glass GE, et al. Longitudinal study of infection with *Borrelia burgdorferi* in a population of *Peromyscus leucopus* at a Lyme disease–enzootic site in Maryland. Am J Trop Med Hyg.1999a;60:598–609.
[14] PerOlof Bengtsson, Nico Lassing, Jan Bosch and Hans van Vliet- "Analyzing Software Architectures for Modifiability"
[15] Xulin Zhao, Foutse Khomh, and Ying Zou, "Improving the Modifiability of the Architecture of Business Applications".
[16] F. Bachmann, L. Bass, and R. Nord, Modifiability tactics, Technical report CMU/SEI-2007-TR-002. Software Eng. Inst., 2007.
[17] ISO/IEC 20926, IFPUG Functional Size Measurement Method, http://www.iso.org/iso/ catalogue_detail.htm?csnumber=51717
[18] W. Stevens, G. Myers, and L. Constantine, Structured Design, IBM Systems Journal, vol. 13 (2): 115-139, 1974.
[19] RS. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill Higher Education, 5th edition, 2001.
[20] W. Li and S. Henry, Object-oriented metrics that predict maintainability. Journal of Systems and Software, vo. 23(2): 111–122 1993.
[21] S. Mancoridis, BS. Mitchell , C. Rorres, and Y. Chen, Using Automatic Clustering to Produce High Level System Organizations of Source Code, Proc. the 6th Int. Workshop on Program Comprehension. pp. 45, 1998.
[22] NE. Fenton and A. Melton, Deriving structurally based software metrics, Journal of Systems and Soft., vol. 12 (3): 177-187, 1990.
[23] Object Constraint Language (OCL), http://www.omg.org/spec/ OCL/2.2/PDF

About Authors

Mr. P. Ashok Reddy, has received his Master's Degree from JNTU Kakinda with Computer Science & Engineering specialization and at present he pursuing Doctor of Philosophy from Acharya Nagarjuna University, AP, India in the area of Software Engineering. He has been actively involved in teaching and research for the past 10 years and now he is working as Sr. Asst. Professor in LBRCE-Mylavaram, AP, India.

Dr. K Rajasekhara Rao, has received his Doctor of Philosophy in Computer Science & Engineering from Acharya Nagarjuna University, AP, India. He has been actively involved in teaching and research for the past 30 years and now he is working as Director, Usha Rama College of Engineering & Technology, Vijayawada, AP, India. His research interests include: Embedded Systems, Software Engineering, Algorithm Complexity analysis and Parallel Computing. Ten of his scholars have successfully earned their PhD degrees in Computer Science & Engineering domain and 5 more scholars are pursuing their PhD work under his guidance from various Universities.

Dr. M. Babu Reddy, has received his Master's Degree and Doctor of Philosophy from Acharya Nagarjuna University, AP, India with Computer Science & Engineering specialization. He has been actively involved in teaching and research for the past 17 years and now he is working as Asst. Professor of Computer Science, Krishna University, Machilipatnam, AP, India. His research interests include: Machine Learning, Software Engineering, Algorithm Complexity analysis and Data Mining.