Consistency Handling In Distributed Requirement Engineering

Neha Bhatija¹, Anshul Chugh²

¹Assistant Professor, Amity University, Gurgaon, Haryana, India
²Assistant Professor, Department of Computer Applications Giani Zail Singh, PTU Campus Bathinda, Punjab

Abstract—Large software projects are characterized by distributed environments which consists different organizations and geographical locations. The distributed environments typically build multiple overlapping models, represents different perspectives, different versions across time, different development concerns, etc. Keeping track of the relationships between these models, constructing a global view, and managing consistency are major challenge. This study addresses the problem of managing the consistency of a set of distributed and possibly heterogeneous requirements. In order to build such models to check the consistency of a set of distributed heterogeneous requirements. This paper provides a mechanism to bridge the heterogeneity gap, to organize requirements, to find a way of expressing consistency constraints and returns the ambiguity of SRS (i.e. software requirements specification) documents. In this research paper a CCT (i.e. Consistency Checker Tool) is designed to verify whether the requirement specification is consistent or not.[1] The CCT tool is developed by using VB.net software. To achieve the ambiguity of documents, ARM (i.e. Automated Requirement Measurement) tool is used to check the results obtained by CCT are unambiguous or not.[2]

Keywords—Software Engineering, Requirements Engineering, SRS, Inconsistency, CCT tool, ARM tool

I. INTRODUCTION

Support for evolutionary processes is needed at all the stages of software development but especially during Requirements Engineering (RE), since it is the requirements modifications that typically trigger other changes in the software development life cycle [Leandro Loper et al., 2008]. Requirements often evolve because stakeholders generally cannot predict all the ways in which a system can be utilized.[14] The environment where the software is situated frequently changes and so do the software boundaries and business rules governing the utilization of that software. Correspondingly, designs change because the requirements change. Implementation has to be changed because designs evolve and defects have to be fixed. For any software project, its software requirement specification plays an important role in the development process. An SRS is a document in the beginning of software development i.e. the requirement definition phase. If there are some errors in the SRS, these errors may lead new errors in the further phases[17]. Since it takes extra cost and time to interpret such errors in the SRS. The SRS should be correct.

II. PROBLEM DEFINITION

One of the main difficulties imposed by geographically distributed software team is the requirement engineering process. The development of complex software systems is a complex and lengthy activity that involves the participation and collaboration of many stakeholders.[12] This results in many partial models of the developing system. These models can be inconsistent with each other since they describe the system from different perspectives and reflect the views of the stakeholders involved in their construction. Developing a desirable framework for handling inconsistencies in software requirements specifications is a challenging problem[3]. It has been recognized that the relative priority of requirements can help developers to make some necessary decisions for resolving conflicts.

Objective of Study

The main objective of this study is:

1. To make a consistent software requirement specification so that efficient and effective software can be developed[5].
2. Designing a tool that can check whether there is consistency in all requirements that are taken from different users or not[4].
3. To diagnose and locate the inconsistency in requirements.

III. RELATED WORK

The purpose of studying relevant work is to understand various ways that can be used to understand how to handle inconsistency in distributed software requirement specifications[8]. Schwanke and Kaiser [R.W Schwanke et.al, 1988] propose an approach to “living with inconsistency in large system”, implemented in the CONMAN programming environment. CONMAN handles inconsistency by: identifying and tracking the six different kinds of inconsistencies; reducing the cost of restoring type safety after a change and protecting programmers from inconsistent code[9].

© 2014, IJARCSSE All Rights Reserved
Bhateja et al., International Journal of Advanced Research in Computer Science and Software Engineering 4(12), December - 2014, pp. 767-771

Balzer [R.Blazer, 1991] proposed the notion of “tolerating inconsistency” by relaxing consistency constraints during development. The approach suggests that inconsistent data be marked by guards (“pollution markers”) that have to identify the inconsistent data to code segments or human agents that may then help resolve the inconsistency, and to screen the inconsistent data from other segments that are sensitive to the inconsistencies[15].

Gabbay and Hunter [D.Gabby et.al., 1992] suggested “making inconsistency respectable” by proposing that inconsistencies be viewed as signals to take external actions, or as signals for taking internal actions that activate or deactivate other rules. Gabbay and Hunter further proposed the use of temporal logic to specify these meta-level rules.

Narayanaswamy and Goldman [K.Narayanaswamy et.al., 1992] proposed “lazy consistency”. The approach is a compromise between the optimistic view in which inconsistencies are assumed to occur infrequently and can thus be handled individually when they arise, and a pessimistic approach in which inconsistencies are prevented from ever occurring. Such a compromise is particularly realistic in a distributed development setting where conflicts or “collisions” of changes made by different developers may occur[10]. Lazy consistency maintenance supports activities such as negotiation and other organizational protocols that support the resolution of conflicts and collisions.

Michael Goedicke, Torsten Meyer, and Christian Piewetz [M.Goedicke et.al., 1998] given an approach “On Detecting and Handling Inconsistencies in Integrating Software Architecture: Design and Performance Evaluation” that based on organizing the entire development process using hierarchically organized ViewPoints[13]. They used an extended version of the ViewPoints framework to model the interaction of two cooperating engineering approaches for designing the architecture of distributed systems.

John Grundy, John Hosking, Member and Warwick B. Mugridge in “Inconsistency Management for Multiple-View Software Development Environments” [J.Grundy, J.Hosking, 1998] had developed a new software architecture, Change Propagation and Response Graphs (CPRGs), for managing inconsistencies in multiple view software development tools[17]. They had developed a software architecture which represents inconsistencies as “change description” objects. Change descriptions are propagated between multiple-view representations with inconsistencies being detected if structure changes cannot be automatically applied by environments, semantic constraints are violated, or multiple developer interference occurs.

William N. Robinson, Member and Suzanne D. Pawlowski, “Managing Requirements Inconsistency with Development Goal Monitors” [William N. Robinson et al., 1999], presented a technique, development model, called a requirements dialog meta-model that that manage requirements document inconsistency by managing inconsistencies that arise between requirement development goals and requirements development enactment,. This meta-model defines a conceptual framework for dialog goal definition, monitoring, and in the case of goal failure, dialog goal reestablishment[19]. The requirements dialog meta-model is supported in an automated multiuser World Wide Web environment, called DEALSCRIBE.

Bashar Nuseibeh, Steve Easterbrook, Alessandra Russo developed the framework in “Leveraging Inconsistency in Software Development” [B.Nuseibeh et al., 2000] that can explicit use of a set of consistency rules, which provide a basis for most inconsistency management activities. The consistency rules are used to monitor an evolving set of descriptions for inconsistencies [20]. When inconsistencies are detected, some diagnosis is performed to locate and identify the cause. At this point, you would choose from among several different inconsistency-handling strategies, including resolving the inconsistency immediately, ignoring it completely, or tolerating it for a while[18]. Whatever action you choose, the result needs to be monitored for undesirable consequences.

Kedian Mu, Weiru Liu, Zhi Jin, Ruqian Lu, Anbu Yue given “A Merging-Based Approach to Handling Inconsistency in Locally Prioritized Software Requirements”. [K.Mu et al., 2007] In this paper, each viewpoint involved in the inconsistency is transformed into a stratified knowledge base, whilst the relationship between these viewpoints is considered as an integrity constraint. Based on the merging operators presented in, they construct a stratified merged knowledge base as an overall view of these inconsistent viewpoints[22]. The ordering relationship over this stratified merged knowledge base could be considered as a global prioritization over the requirements specification. Generally, the requirements with lower merged preference may be considered as requirements to be abandoned. Then we may derive some proposals for handling the inconsistency from the merged result[14]. The global prioritization as well as the local prioritization may be used to argue these proposals and help developers identifying acceptable common proposals[24].

Kedian Mu, Weiru Liu, Zhi Jin and David Bell in “Handling Inconsistency In Distributed Software Requirements Specifications Based On Prioritized Merging” [K.Mu et al., 2009] present a prioritized merging-based framework for handling inconsistency in distributed software requirements specifications. Given a set of distributed inconsistent requirements collections with the local prioritization, they first construct a requirements specification with a prioritization from an overall perspective[27]. They provide two approaches to constructing a requirements specification with the global prioritization, including a merging-based construction and a priority vector-based construction. Following this, they derive proposals for handling inconsistencies from the globally prioritized requirements specification in terms of prioritized merging[31]. Moreover, from the overall perspective, these proposals may be viewed as the most appropriate to modifying the given inconsistent requirements specification in the sense of the ordering relation over all the consistent subsets of the requirements specification.

IV. EXPERIMENTAL WORK

In this research, developed a Consistency checker Tool (CCT). The objective of the Consistency checker Tool (CCT) is to check the correctness of the software requirement specification. The correctness means to check that the requirement specification is consistent or it holds inconsistent requirements.

© 2014, IJARCSSE All Rights Reserved
The CCT tool scans the values designated by the user that contains the requirement specifications. During the scanning process, the CCT tool matches each requirement value given by users to the values that are already stored in database[24]. The approach to checking consistency presented in this research has its root in “Leveraging inconsistency in Software Development” [B.Nuseibeh et al., 2000]. The tool provides a way to specify constraints that express relationship between elements in multiple documents, regardless of their storage location. To check whether the requirement specification is unambiguous or not we use the Automated Requirement Measurement Tool [ARM][27]. The objective of the Automated Requirement Measurement Tool (ARM) is to provide measures that can be used to assess the quality of a requirement specification document. The tool is not intended to evaluate the correctness of the specified requirements. It is an aid to “writing the requirements right” not “writing the right requirements”. The ARM tool scans a file designated by the user as the text file that contains the requirement specifications. During the scanning process, the ARM tool searches each line of text for specific words and phrases. These search arguments (specific words and phrases) are considered to be indicators of the document’s quality as a specification of requirements. While the source file is being scanned, an ASCII text report file is created in the same directory as the source “.txt” file designated by the user. The ARM tool gives the report file the same prefix name as the user’s source file and adds an extension of “arm”[28]. This report file contains a summary report and detailed reports for specific quality indicators.

V. RESULT

The results of CCT tool are based on comparing all the requirements specifications of all different users with the already specified requirements[30].

![Figure 1: 1st step of Comparision Form](image1)

For this choose the project name then its corresponding module name and then user name. After click on check button the result of comparisons of requirements will shown in data grid view of the form.

![Figure 2: Results of Comparisons](image2)

In the above diagram, yellow color indicate that requirement that is not fill by the user.

![Figure 3: Another Result of Comparision](image3)

If inconsistency occurs in any requirement then tool highlighted that row with red color. If tool find a consistent record then it highlight it with green color. By this we can find whether inconsistency occurs in requirement specification or not. After finding inconsistency records by consistency checker tool, next step is to generate the report according to the analysis in natural language[29].
Figure 4: Results after use of CCT

To check whether generated report is unambiguous or not, it can be analyze by ARM tool. The user can run the requirement specification document through the ARM tool, to do this select the Analyze Document button.

Figure 5: Shows Result of all occurrences

VI.  CONCLUSIONS AND FUTURE WORK

The software requirement specification is valid if it is consistent[14]. To make a consistent specification, it is necessary to handle the inconsistencies that occur in the requirement specifications. Identifying appropriate actions or proposals for handling inconsistency is still a big problem in requirements engineering. During research, Consistency Checker Tool is designed that can help the developer team to make a consistent software specification. The developed tool is tested on standalone system by taking the requirements from different users. It can be made distributed by connecting more systems located at geographical locations so that requirement specification can be checked. If there is inconsistency occur in requirement specification it can be easily detected. The tool can help to find out the inconsistent requirements by changing the colors of those requirements. So by this, it is easily find out that which requirement is consistent or not and a report can be prepared about inconsistent requirement[20].

But the main problem that still occurs is that inconsistency can be easily detected but it cannot be find out that the risk of that inconsistent requirement is up to what level. The consistency checker tool only detects the inconsistency, not how to handle or remove that inconsistency[11]. To enhance the designed consistency checker tool so that tool can also help us to find out the risk level of inconsistencies. Because according to these risk levels, further actions can be taken. For example if the risk level of inconsistency is low then it can be ignored or some techniques can be used to manage it. But if the risk level is high then some techniques can be used to handle it or remove the inconsistencies.

REFERENCES


[31] William N.Robinson and Suzanne D.Pawlowski:” Managing Requirements Inconsistency with Development Goal Monitors”, Member, IEEE, 0.98-5589, 1999