© 2014 IJEDR | Volume 2, Issue 2 | ISSN: 2321-9939

Inconsistency Measurement and Remove from Software Requirement Specification

[1]karuna patel, [2]Prof.Snehal Gandhi
[1]Computer Department
[1]Sarvajanik College of Engineering and Technology, Surat, Gujarat, India
[1]karoona11@yahoo.com, [2]snehal.gandhi@scet.ac.in

***Abstract* - Removing inconsistency in software requirements is a complicated task which has attracted the interest of many groups of researchers. Formal and semi-formal specifications often have inconsistencies in the depicted requirements that need to be managed and resolved. This is particularly challenging when refining informal to formalized requirements. Our aim is to better support users and developers to work with informal and semi-formal requirements and keep them consistent. We will assist requirement engineers and business analysts to check whether their requirements that are written or collected in natural language are consistent with other analysis and design representation. By improving method of inconsistency we can remove the inconsistency and because of this method we can improve the correctness of the software requirements as well as we will improve the quality of software requirement specification.**

***Index Terms* - Ontology, natural language processing, inconsistency, First-order logic**

I. INTRODUCTION

Ontology is originally a branch of philosophy, about the basic characteristics of all reality in the world. Currently, it has become a popular topic in various communities, including artificial intelligence, knowledge engineering, and natural language processing. Different definitions are adopted for each community. The commonly accepted definition is "ontology is a formal, explicit specification of a shared conceptualization". There is much well-known ontology developed by different communities, such as WorldNet, Cyc, etc. They are categorized into different types with respect to different criteria, and implemented in different forms with different models. There are many issues along with ontological research, and ontology construction is the most important one. The associated study has been conducted for nearly twenty years [8].

What Is Inconsistency In Ontology?

We use the term inconsistency to denote any situation in which a set of descriptions does not obey some relationship that should hold between them. The relationship between descriptions can be expressed as a consistency rule against which the descriptions can be checked. In current practice, some rules may be captured in descriptions of the development process; others may be embedded in development tools. However, the majority these type of rules are not captured anywhere. [3]

Theoretical Back ground and Literature Survey

There are two ways that we can measure inconsistency of software requirements specifications, qualitative analysis and quantitative analysis. Qualitative analyses classify inconsistency by the conditions why inconsistency occurs, while quantitative analysis calculates severe degree of inconsistency. In the qualitative analysis of requirements inconsistency, B.Nuseibeh [1] proposes a framework which has been widely accepted in the community of requirements engineering. V.Lamsweerde [7] gives classification of inconsistency by the set of assertions, boundary conditions, and the number of assertions in the requirements specifications. Thus inconsistency can be categorized as conflict, obstacle, divergence and obstruction. [2]

First-order logic

First-order logic is a formal logical system that expands propositional logic by additional logical operators of quantifiers [8]. It is built around objects and relations, and provides a well-defined knowledge representation that is both amenable to automated reasoning and relatively easy for analysts to understand. FOL has been applied for ontology modeling. For example, the Knowledge Interchange Format (KIF), built on FOL, was proposed for ontology modeling. Recently logic representations with limited expressiveness, such as description logics, have been widely used, but extra logical assumptions are necessary to capture the intended semantics of the terms in ontology.

Semantic Networks

A semantic network is the graph of the structure of meaning. Specifically, "it is a graphical notation for representing knowledge in patterns of interconnected nodes and arcs". The nodes represent the concepts and the arcs are the interrelationship between every two nodes. It provides a convenient approach to visualize a knowledge base. Semantic network has been applied for many ontology development projects. It is believed that semantic network is the most appropriate representation method for capturing and encapsulating the massive amounts of semantic information in an intelligent environment. [8]

Causes of Inconsistencies (the why)

Inconsistency is an unavoidable part of software development processes. Even in the most well-defined, managed and optimized development process, system requirements are often uncertain or contradictory, alternative design solutions exist, and errors in implementation arise. The requirements engineering stage of development is particularly illustrative of such inconsistencies. During requirements acquisition, customer requirements are often sketchy and uncertain. For large projects in particular, a number of "client authorities" may exist who have conflicting, even contradictory requirements. [4]

Checking consistency rules

Here there are three examples of consistency rules expressed in English:

1. In a dataflow diagram, if a process is decomposed in a separate diagram, the input flows to the parent process must be the same as the input flows to the child dataflow diagram.
2. For a particular library system, the concept of an operations document states that and borrower are synonyms. Hence, the list of user actions described in the help manuals must correspond to the list of borrower actions in the requirements specification.
3. Coding should not begin until the Systems Requirement Specification has been signed off by the project review board. Hence, the program code repository should be empty until the status of the SRS is changed to "confirmed". The first rule applies to two descriptions written in the same notation. The second rule describes a consistency relationship that must be maintained between three different documents. The third rule expresses a relationship between the statuses of two descriptions to ensure that they are consistent with the stated development process. [3]

Detecting &Identifying Inconsistencies (the how)

Once consistency has been defined in terms of explicit rules, inconsistency may be detected automatically by checking these rules. For example, a type checker can check whether or not an instance or variable conforms to its type definition. Similarly, a parser can check whether or not a sentence conforms to the syntactic rules specified by its grammar. Simple inferences in classical logic can also be used to detect logical inconsistencies resulting from too much or too little information. For example, a contradiction (the simultaneous assertion, or deduction, of both X and ¬X) may be detected in this way. Other kinds of inconsistency are more difficult to detect. [4]

Acting in the presence of inconsistency (the what)

The approaches described above handle inconsistencies in different ways. What they have in common, however, is the goal of allowing continued development in the presence of inconsistency. Such action may include:

Ignoring the inconsistency completely and continuing development regardless. This may be appropriate in certain circumstances where the inconsistency is isolated and does not affect further development, or prevent it from taking place. [4]

Handling inconsistency

The choice of an inconsistency-handling strategy depends on the context and the impact it has on other aspects of the development process. Resolving the inconsistency may be as simple as adding or deleting information from a software description. But it often relies on resolving fundamental conflicts or making important design decisions. In such cases, immediate resolution is not the best option. You can ignore, defer, circumvent, or ameliorate the inconsistency. Sometimes the effort to fix an inconsistency is significantly greater than the risk that the inconsistency will have any adverse consequences. In such cases, you may choose to ignore the inconsistency. Good practice dictates that such decisions should be revisited as a project progresses or as a system evolves. Deferring the decision until later may provide you with more time to elicit further information to facilitate resolution or to render the inconsistency unimportant. In such cases are flagging the affected parts of the descriptions is important. [4]

Measuring inconsistency

For several reasons, measurement is central to effective inconsistency management. Developers often need to know the number and severity of inconsistencies in their descriptions, and how different changes that they make affect these measures. Developers may also use these measures to compare descriptions and assess, given a choice, which is preferred. Often developers need to prioritize inconsistencies in various ways to identify inconsistencies that need urgent attention. They may also need to assess their progress by measuring their conformance to some predefined development standard or process model. The actions taken to handle inconsistency often depend on an assessment of the impact these actions have on the development project. Measuring the impact of inconsistency-handling actions is therefore a key to effective action in the presence of inconsistency. You also need to assess the risks involved in either leaving an inconsistency or handling it in a special way. [4]

II. PROPOSED WORK

Improve the inconsistency method

In this method we have to improve the inconsistency which is occurring in software requirement development. By improving inconsistency can maintain but the software requirement specification needs completeness and correctness will diminish correctness. This drawback will improve by this method and completeness can also be maintained. In our proposed work we are going to make algorithm that can be remove the inconsistency from the software requirement specification ontology.

III. IMPLEMENTS AND RESULTS

Here we have developed the ontology of Library Management System and then read the software requirement specification. Checking the rules if any rule is broken down then we can say that inconsistency is arise in software requirement specification or any contradictions arise in document. They are listed in figure below. After that we have remove the inconsistency from the software requirement specification.

Steps for check and remove inconsistencies

- Make ontology of Software requirement specification.
- Checking rules with whole document.
- If any inconsistency is detected,
- Save in temporary file ,
- Then take a link with document and remove it from document.

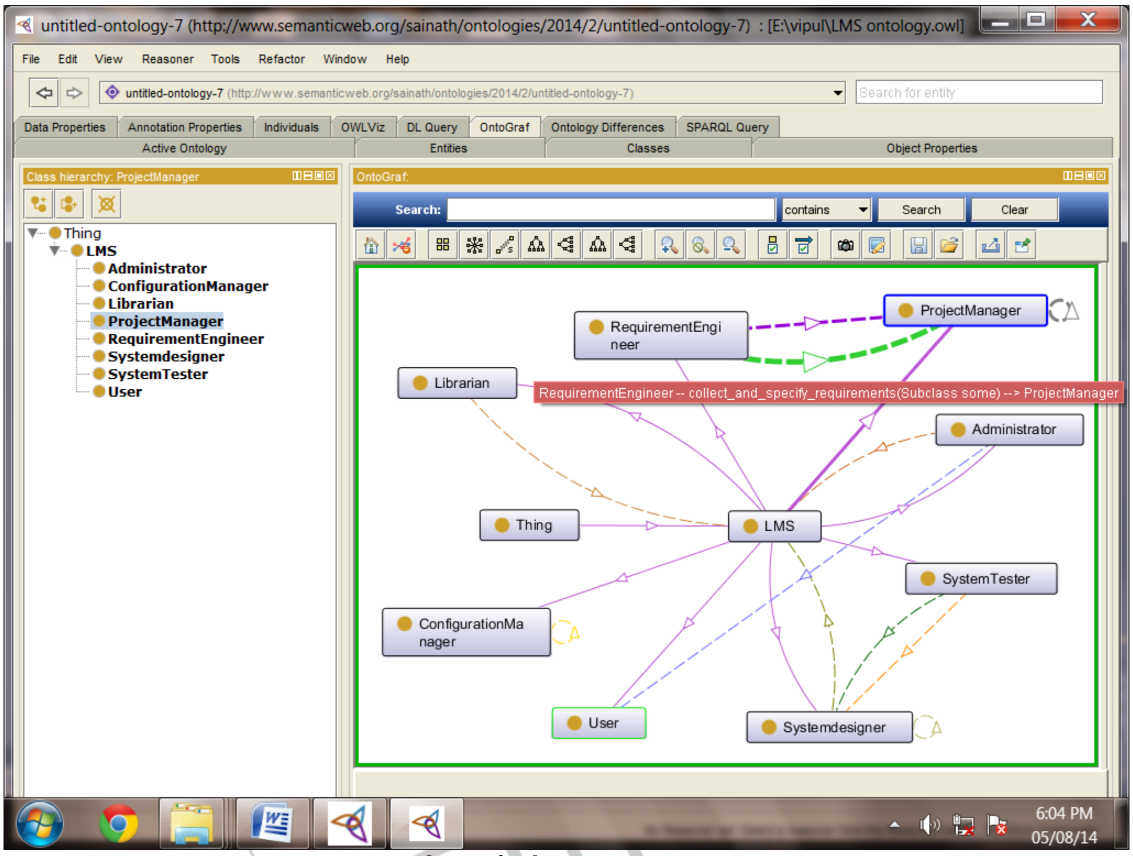
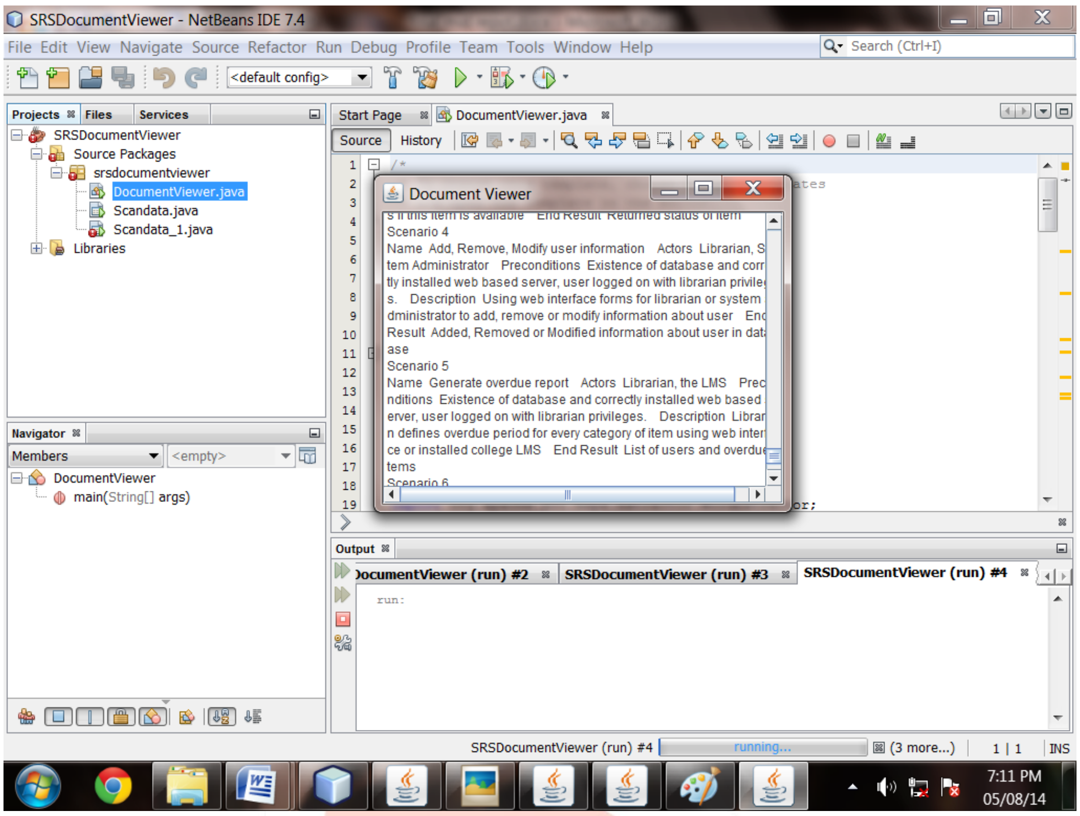


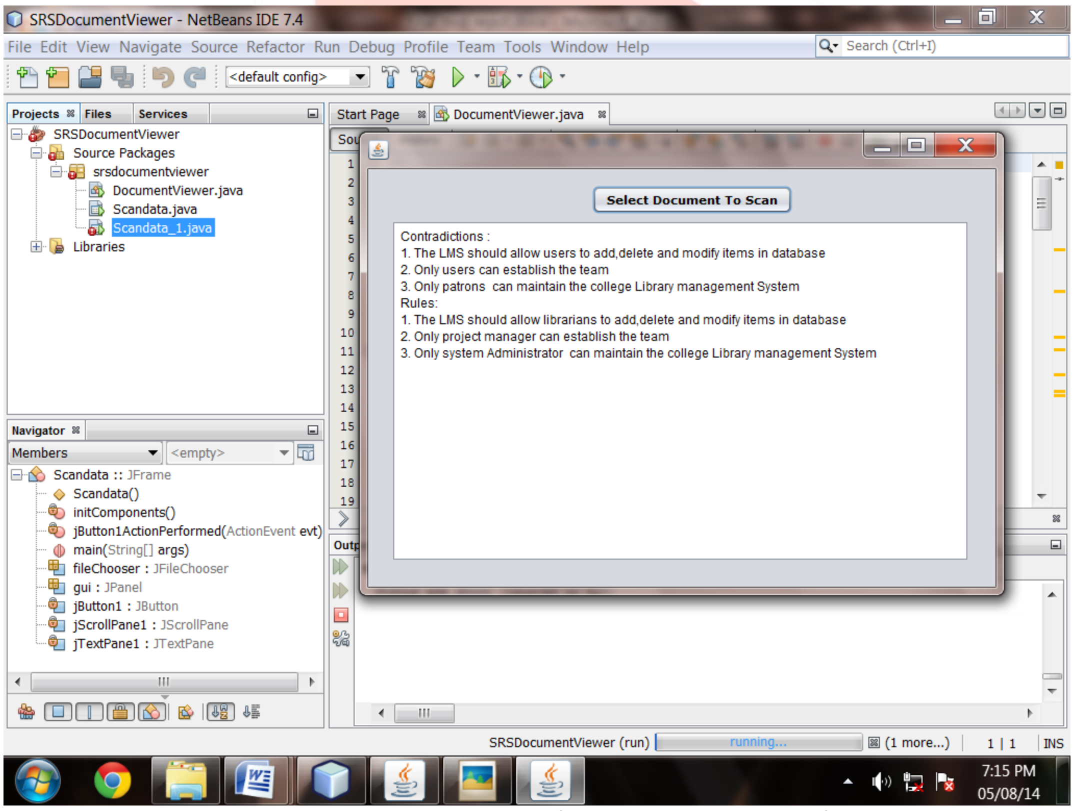
Fig 1 ontology of Library Management system



```
s if this item is available   End Result  Returned status of item
Scenario 4
Name  Add, Remove, Modify user information   Actors  Librarian, S
tem Administrator   Preconditions  Existence of database and corr
tly installed web based server, user logged on with librarian privile
s.   Description  Using web interface forms for librarian or system
dministrator to add, remove or modify information about user   End
Result  Added, Removed or Modified information about user in dat
ase
Scenario 5
Name  Generate overdue report   Actors  Librarian, the LMS   Prec
nditions  Existence of database and correctly installed web based
erver, user logged on with librarian privileges.   Description  Librar
n defines overdue period for every category of item using web inter
ce or installed college LMS   End Result  List of users and overdue
tems
Scenario 6
```

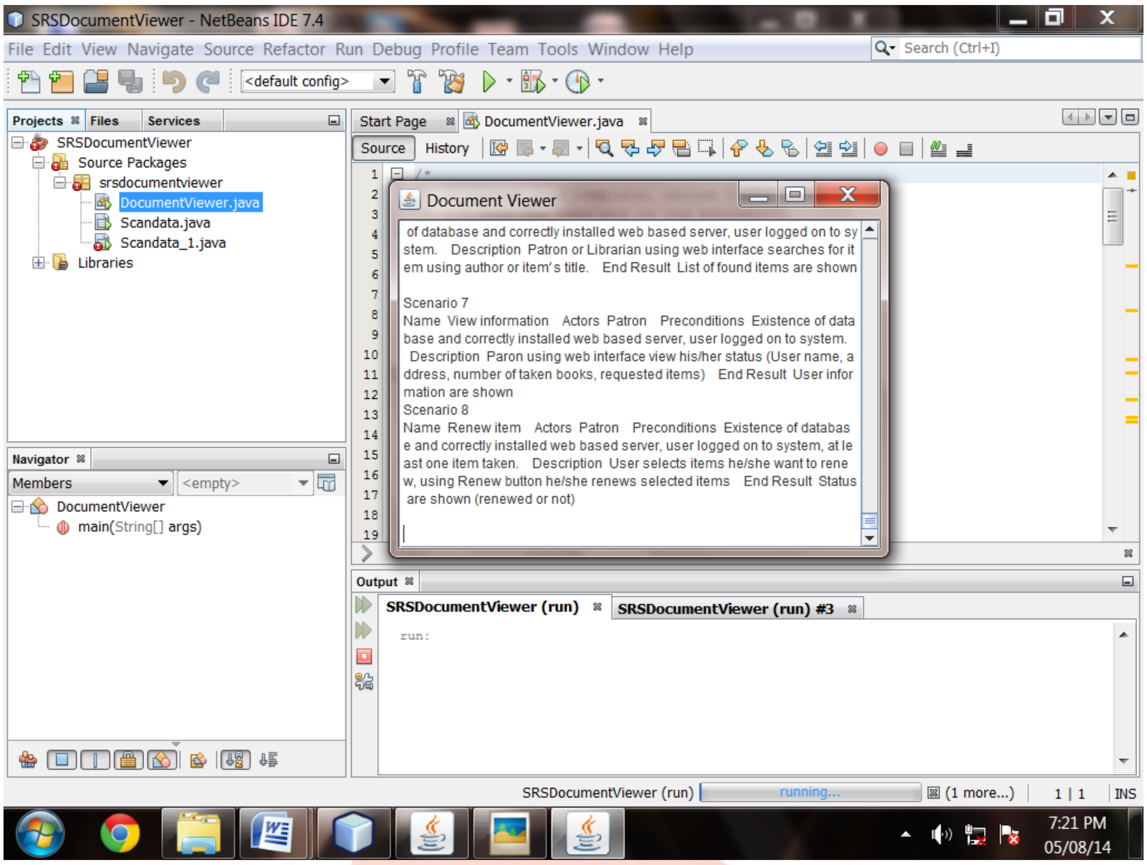
```
run:
```

Fig 2 Read the srs (software requirement specification) document



```
Contradictions :
1. The LMS should allow users to add,delete and modify items in database
2. Only users can establish the team
3. Only patrons  can maintain the college Library management System
Rules:
1. The LMS should allow librarians to add,delete and modify items in database
2. Only project manager can establish the team
3. Only system Administrator  can maintain the college Library management System
```

Fig 3 Detecting contradictions in software requirement specification



```
 of database and correctly installed web based server, user logged on to sy
stem.   Description  Patron or Librarian using web interface searches for it
em using author or item's title.   End Result  List of found items are shown

Scenario 7
Name  View information   Actors  Patron   Preconditions  Existence of data
base and correctly installed web based server, user logged on to system.
 Description  Paron using web interface view his/her status (User name, a
ddress, number of taken books, requested items)   End Result  User infor
mation are shown
Scenario 8
Name  Renew item   Actors  Patron   Preconditions  Existence of databas
e and correctly installed web based server, user logged on to system, at le
ast one item taken.   Description  User selects items he/she want to rene
w, using Renew button he/she renews selected items   End Result  Status
 are shown (renewed or not)
```

```
run:
```

Fig 4 removes the contradictions from srs.

IV. Conclusion

In software requirement specification there are various rules are defined and when the rule is broken down inconsistency arise in srs (software requirement specification) because of some contradictions. Here we have tried to remove the contradictions from software requirement specification and we can achieve the correctness of software requirement specification. This way we can improve the quality of the software requirement specification.

References

[1] Massila Kamalrudin,"Automated Software Tool Support for Checking the Inconsistency of Requirements" University of Auckland, Private Bag International Conference on Automated Software Engineering, IEEE/ACM, 2009.
[2] Xuefeng Zhu1,2,Zhi Jin1,3 "Inconsistency Measurement of Software Requirements Specifications: An Ontology-Based Approach" Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing, IEEE,2005.
[3] Bashar Nuseibeh, Steve Easterbrook, Alessandra Russ "Leveraging Inconsistency in Software Development"IEEE, 2000.
[4] Bashar Nuseibeh,"To Be and Not to Be: On Managing Inconsistency in Software Development", London, IEEE, 2001.
[5] Egyed, A., "Scalable Consistency Checking Between Diagrams-The ViewIntegra Approach", in Proceedings of the 16th IEEE international conference on Automated software engineering, IEEE, 2001.
[6] Denger, C., D.M. Berry, and E. Kamsties, "Higher Quality Requirements Specifications through Natural Language Patterns", in Proceedings of the IEEE International Conference on Software- Science, Technology \& Engineering, IEEE,2003.
[7] V.Lamsweerde, R.Darimont, E.Letier. "Managing Conflict in Goal-Driven Requirements Engineering",Transactions on Software Engineering, Vol.24,IEEE, 1998.
[8] J. M. Park, J. H. Nam, Q. P. Hu, H. W. Suh, "Product Ontology Construction from Engineering Documents", Yusong-ku, Taejon, 305-701 Republic of Korea, International Conference on Smart Manufacturing Application,2008.