Managing Consistency between Textual Requirements, Abstract Interactions and Essential Use Cases

Massila Kamalrudin
Electrical & Computer Engineering
University of Auckland
Private bag 92019
Auckland 1142
New Zealand
Mkam032@aucklanduni.ac.nz

John Grundy
Swinburne University of Technology
Centre for Complex Software, Systems & Services, PO Box 218, Hawthorn
Victoria 3122, Australia
jgrundy@swin.edu.au

John Hosking
Dept of Computer Science
University of Auckland
Private bag 92019 Auckland 1142
New Zealand
john@cs.auckland.ac.nz

Abstract— **Consistency checking needs to be done from the earliest phase of requirements capture as requirements captured by requirement engineers are often vague, error-prone and inconsistent with users' needs. To improve such consistency checking we have applied a traceability approach with visualization capability. We have embedded this into a light-weight automated tracing tool in order to allow users to capture their requirements and generate Essential Use Case models of these requirements automatically. Our tool supports inconsistency checking between textual requirements, abstract interactions that derive from the text and Essential Use Case models. A preliminary evaluation has been conducted with target end users and the tool usefulness and ease of use are evaluated. We describe our motivation for this research, our prototype tool and results of our evaluation.**

Keywords- Automated Tracing Tool, Traceability, Essential Use Cases, Consistency management, Inconsistency

I. INTRODUCTION

The requirements of a system to be developed need to be evaluated against the three Cs (Consistency, Completeness and Correctness) to detect errors such as inconsistency and incompleteness. However, as stated by Zowghi and Gervasi, "*improving the consistency of the requirements can reduce the completeness and, thereby again diminishing correctness"* [1]. Therefore, consistency is of great interest here in order to ensure the requirements are entirely precise and fulfill the needs of a user. In order to make sure requirements are consistent and follow the customers' needs from the beginning, consistency checking needs to be done from the earliest stage of the Requirement Engineering process: Requirements Analysis (RA).

There are several definitions of consistency with respect to a software requirement specification. These definitions clarify what consistency is and when it appears in a software requirement specification. Zowgi and Gervasi [1] state that consistency requires that no two or more requirements in a specification contradict each other, where there is no case that the requirement cannot be compensated at the same time. They also stress the importance of terminology i.e. that words and terms always having the same meaning throughout the requirement specification. Both of these views entail the need for ways of avoiding reciprocally exclusive statements and conflicts in terminology [2]. Consistency is referred to as no internal (logical) negation between specification of a system [3]. A few types of consistency apply to specifications including the precondition of a function being satisfied by the function calls, subtypes that inclusive arguments of functions, and results of functions subtypes [3]. Some relate to consistency between various non-functional requirements e.g. that security, reliability, scalability and platform requirements can all be met by the requirements as captured.

In order to check and maintain consistency and diminish inconsistency, many techniques have been used. These include traceability, formal specifications, semantics analysis, semi-formal specifications and heuristic algorithms[4],[5],[6],[7],[8]. In addition, in many projects consistency and completeness checking is normally performed manually by "*tedious procedure of reading the requirements documents and looking for linguistic errors*" [9]. Many of these approaches to requirements consistency checking require heavy-weight formal approaches where requirements must be expressed in complex formal models. While these are important in many domains e.g. safety-critical systems, they have proved challenging to put into widespread use [10]. Similarly traditional approaches to using natural language processing and analysis of textually expressed requirements require the use of complex analysis algorithms and the complexity of natural language and its inherent ability to express inconsistent statements makes this challenging [11]. Translating requirements into semi-formal models e.g. UML use cases is a common approach that supports some limited analysis while improving structuring of natural language expressed requirements. However, carrying out the translation to these semi-formal models and checking consistency between the models and natural language requirements has continued to prove problematic [12]. We wanted to provide requirements engineers with an environment to support consistency checking and traceability between semi-formal models of requirements and natural language expressed requirements [13]. However, we wanted to provide requirements engineers with a more light-weight approach than complex natural language processing techniques or their having to rely on using complex, mathematical formal models. To this end we have developed a prototype tool supporting the extraction of Essential Use Case (EUC) models from natural language requirements and support for traceability and consistency management between these requirements models. Our toolset is built in the Eclipse IDE and allows requirements engineers, end users, and other developers to work with both textual natural language and diagrammatic EUC models of requirements.

This paper is organized as follows. We begin with an introduction to the concept of consistency in software requirements specification and follow with the background and our motivation of traceability between textual and Essential Use Case diagram requirements in Section 2. Section 3 illustrates the approach we have taken and in section 4 we describe our prototype automated tracing tool by use of an example. Section 5 discusses the architecture and the implementation of our tool and section 6 discusses an evaluation of the tool. Section 7 compares and contrasts our approach to key related work and section 8 presents conclusions and directions for future research.

II. BACKGROUND AND MOTIVATION

A. Traceability

Traceability is defined as the "ability to describe and follow the life of an artefact which is developed during software lifecycle in both forward and backwards directions' [14]. Traceability is believed to be an important approach in managing requirements effectively [15] and a vital practise in an organisation [16]. Traceability must also cover all the aspects in terms of scope and coverage including system level scope and all four types of coverage. The four types of coverage defined by Bashir et al [17] are the traceability of origin and requirement inclusive source, stakeholders and requirements. Next, all requirements are involved in traceability between requirements and other requirements. Different requirements are also traced if they are dealing with the traceability between requirements and other artefacts, and links and dependencies between artefacts need to be considered if we are tracing other artefacts with other artefacts.

Cysneiros and Zisman assert that traceability relations help in a number of activities in software development [18]. For example the evolution of software systems, compliance verification of code, requirements validation, aspect identification, and any design decision. Traceability is often informally practised in tracing requirements to and from a software design [16]. Some traceability techniques are assisted by information retrieval (IR) to support identifying traceability links although IR is unable to identify all links [14, 18]. Although traceability is important it is sometimes not applied in practice as it is too difficult and costly [16]. In this paper we present an approach which applies traceability together with Essential use cases (EUC) in an automated tracing tool, in order to demonstrate that traceability can be easy to use and that benefits accrue from being able to trace between different forms of requirements and check requirements consistency.

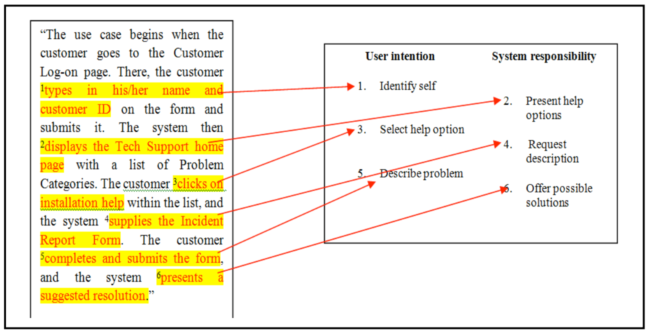
B. Consistency

Consistency management between different artefacts in software engineering has been recognized as crucial for many years [5],[19],[20]. In requirements engineering, consistency management between formal requirements specifications and architecture and design models has been investigated [7],[4]. Similarly, several approaches have been developed to try and determine inconsistencies between natural language descriptions of requirements and formalized models of requirements [5],[9]. Some techniques have been developed to support correction of inconsistencies such as the use of repair operations [21]. Detecting inconsistencies may or may not require immediate correction. Living with inconsistency allows for the management of inconsistencies over time where this provides more flexibility in the development process [22]. Correcting inconsistencies and providing appropriate tool support to detect, present and manage is challenging [23].

C. Essential Use Cases (EUCs)

In previous work we have developed tools to support traceability, inconsistency detection and consistency management between different semi-formal and formal models of architectures, designs, code and tests [23],[24]. However we did not address the issue of traceability or consistency management of textual, natural language requirements and semi-formal or formal models of requirements. To explore this domain we needed a formal or semi-formal model to represent requirements derived from textual, natural language requirements.

An Essential Use Case (EUC) is defined as a "*structured narrative, expressed in a language of the application domain and of users, comprising a simplified, generalized, abstract, technology free and independent description of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction*" [25]. An EUC is shorter and simpler than conventional use cases, and is in the form of a dialogue between the user and system which helps to support better communication between developers and stakeholders. This technology-free approach assists better requirements capture as it only allows specific detail relevant to the design to be captured [26]. An EUC specifies a sequence of abstract steps and captures the core part of a requirement [26]. It contains user intentions and system responsibilities allowing documentation of the interaction without the need to describe the user interface in detail. The concept of responsibility in EUC is aimed at identifying "*what the system must do to support the use case"* without being concerned about *"how it should be done"*[26]. This concept allows consistency with the role of responsibility in the design. In addition, using responsibilities in EUCs permits profitable research on the consistency issue between the requirement and the design and helps to improve traceability [26]. Figure 1.0 is the example of an EUC designed by [27] together with the extraction of a natural language requirement to the Essential Use Case.



"The use case begins when the customer goes to the Customer Log-on page. There, the customer [1]types in his/her name and customer ID on the form and submits it. The system then [2]displays the Tech Support home page with a list of Problem Categories. The customer [3]clicks on installation help within the list, and the system [4]supplies the Incident Report Form. The customer [5]completes and submits the form, and the system [6]presents a suggested resolution."

Figure 1. An example of an Essential use case (from [28])

Figure 1 shows an example of capturing requirements from the natural language requirement (left hand side) and an example of Essential Use Case (right hand side). The example of the requirement is from [28]. On the left is the natural language requirement and the important phrases are extracted (highlighted). A specific key phrase (essential requirement) is provided based on the extracted phrases and is shown in the Essential Use case on the right.

We have developed an automated tracing tool [13] to extract essential interactions from natural language expressed textual requirements. We have analyzed these textual requirements with an interaction pattern library and a tracing engine to provide a set of EUC abstract interactions automatically. We wanted to extend this work by providing requirements engineers a diagrammatic essential use case model of these essential interactions expressed in the source textual natural language requirements. We wanted to provide interactive trace-forward and trace-back support allowing engineers to move between these different forms of natural language and semi-formal requirements. We wanted to support consistency management between the different forms of the requirements to aid engineers in reviewing and modifying them and keeping them consistent.

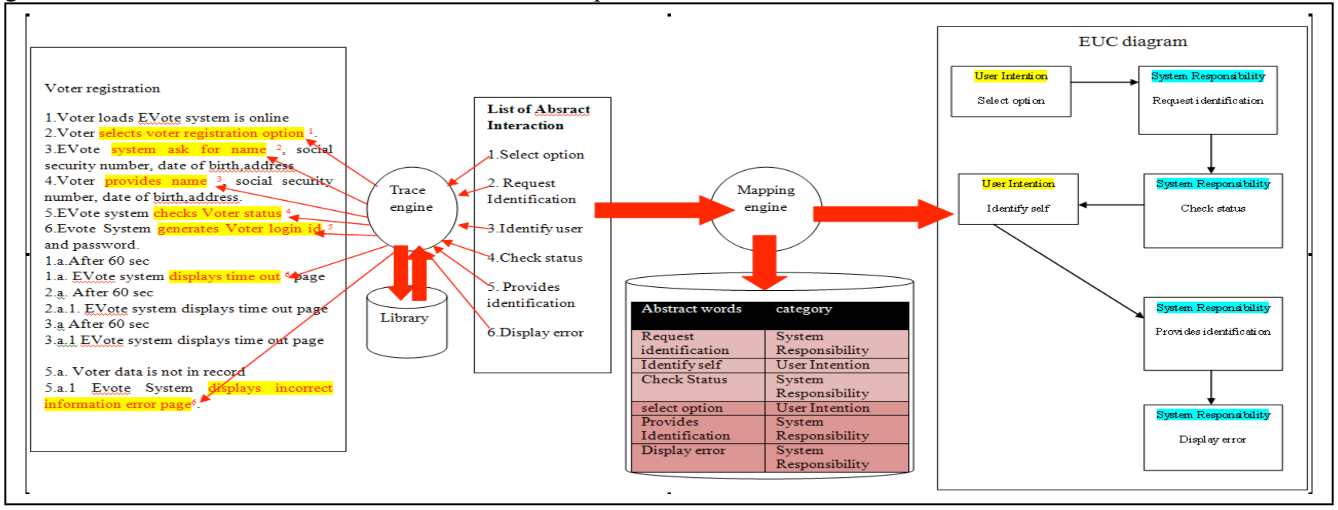
III. OUR APPROACH

We have applied traceability techniques to help support consistency management between textual requirements and EUCs. This work focuses on managing the essential interaction requirements which mainly capture the functional requirements of a system. We have created an "essential interaction" phrase library from the collection and categorization of requirements from different domains and scenarios. Phrases have been extracted and stored in this library and are used to match against corresponding phrases in textual natural language requirements. The extracted phrases are further matched to identify a specific abstract interaction (essential requirement). Each of the abstract interactions is classified as to being a user intention or system responsibility. The derived essential use case elements can be traced back to their originating natural language requirement phrases and vice-versa. We now embed this extraction and tracing support into an Essential Use Case editing tool that we have developed using the Marama meta-tool platform [29] . This now provides an environment in which requirements engineers have the ability to extract and then have generated candidate diagrammatic EUCs automatically from requirements expressed in natural language text. Consistency management support between these textually expressed requirements, a derived set of structured abstract interaction and semi-formal diagrammatic EUCs is then provided. Requirements engineers can move between the different requirements forms using the traceability relationships preserved during the extraction and generation processes. They can modify any one of the requirements forms from the informal natural language text to the semi-formal EUC diagrams and the environment will attempt to update the other forms and/or indicate resultant inconsistencies.

The framework of extracting the requirement, mapping the type of interaction and creating the EUC is shown in Figure 2. Figure 2 (1) illustrates the extraction of a set of abstract interactions from the textual, natural language requirements. The library of abstract interaction phrases is used by a "trace engine" to analyze the text for matches and a set of candidate abstract interactions generated. A "mapping engine" then uses a database of Essential Use Case patterns to structure the interactions into an EUC model (2). The mapping engine then generates a diagrammatic representation of the Essential Use Case (3) which represents the dialogue occurring between the user and system. The traceability relationships between elements in the textual natural language requirements model, the extracted essential interactions model, and the diagrammatic EUC model are preserved and can be used to support traceability between the three forms and to check for inconsistencies between the three forms (e.g. elements in one but not in another; inconsistent naming, ordering or properties of elements; and duplicated or partially duplicated phrases or elements).

IV. TOOL SUPPORT

Based on the framework outlined above we have developed an automated tracing tool, Marama AI, together with an EUC diagram editor, Marama Essential. This work provides support to EUC users and requirements engineers for designing and generating EUCs automatically, minimizing the time to develop them from source textual requirements and increasing the correctness of the abstract interactions produced. In addition this automated tool helps to lessen the need for manual checking of software requirements consistency. The tool provides consistency checking and notification support allowing requirements engineers to modify any of the three forms of requirements in the tool. We used the Marama meta-toolset, a set of Eclipse IDE plug-ins, to develop our Marama AI prototype. Marama AI allows traceability between the textual requirements, abstract interaction and EUCs to be interactively visualized. In addition, any requirements that are incomplete and inconsistent can be highlighted to the user. The tool also comprises a glossary and set of guidelines to assist users to write correct and complete EUC-based requirements.



Voter registration

1.Voter loads EVote system is online
2.Voter selects voter registration option [1].
3.EVote system ask for name [2], social security number, date of birth,address
4.Voter provides name [3], social security number, date of birth,address.
5.EVote system checks Voter status [4]
6.Evote System generates Voter login id [5] and password.
1.a.After 60 sec
1.a. EVote system displays time out [6] page
2.a. After 60 sec
2.a.1. EVote system displays time out page
3.a After 60 sec
3.a.1 EVote system displays time out page

5.a. Voter data is not in record
5.a.1 Evote System displays incorrect information error page[6].

List of Absract Interaction

1.Select option
2. Request Identification
3.Identify user
4.Check status
5. Provides identification
6.Display error

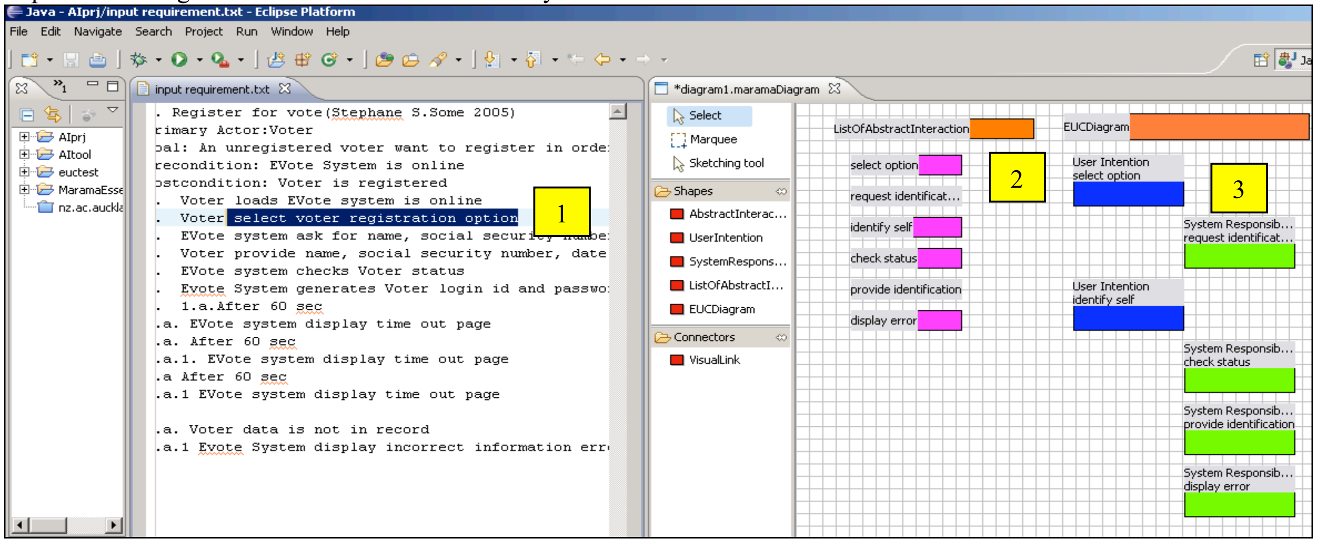
Abstract words	category
Request identification	System Responsibility
Identify self	User Intention
Check Status	System Responsibility
select option	User Intention
Provides Identification	System Responsibility
Display error	System Responsibility

Figure 2. Framework for extracting requirement (1) mapping interactions (2) and creating the EUC automatically (3)

We have conducted a study on the accuracy in terms of correctness of the abstract interactions provided by our automated extraction feature using the interaction patterns provided by the library based on the collection of patterns from Constantine and Lockwood [28], Biddle et al. [30], and patterns developed by us. The results show that the incorrectness and incompleteness of the textual requirement seriously impact the ability to produce correct abstract interactions to structure requirements.

Our automated extraction and tracing tool is shown in Figure 3. Consider the scenario of a voter registration use case by [31] by way of illustration. We use this scenario as a case study to show the benefits and the flow of the consistency checking process. A set of requirements for this voter registration system are expressed in natural language and are open in an Eclipse text editor (1). The natural language requirements do not have to be structured as a list or use a structured layout as shown in this example. The requirements engineer has then had the tool analyze these requirements and a set of "essential interactions" has been deduced from these textual requirements. These essential interactions are then represented as a vertical list (2). Our tracing engine uses a library of phrases and regular expressions to deduce and extract candidate essential interactions. From the essential interaction list extracted our mapping engine generates an EUC diagram (3) using a set of patterns and EUC diagram heuristics. The user can interact with these three representations of requirements: the natural language expressed textual requirements, abstracted essential interactions, and diagrammatic EUC model. One interaction is to select items in one view and see the related items in others i.e. the traceability links.

Figure 3 (1) shows the selected phrase – "select voter registration option" is traced to a particular abstract interaction – "select option" (2). This has then been mapped to the EUC diagram and falls under the "user intention" category (3) and select option interaction.



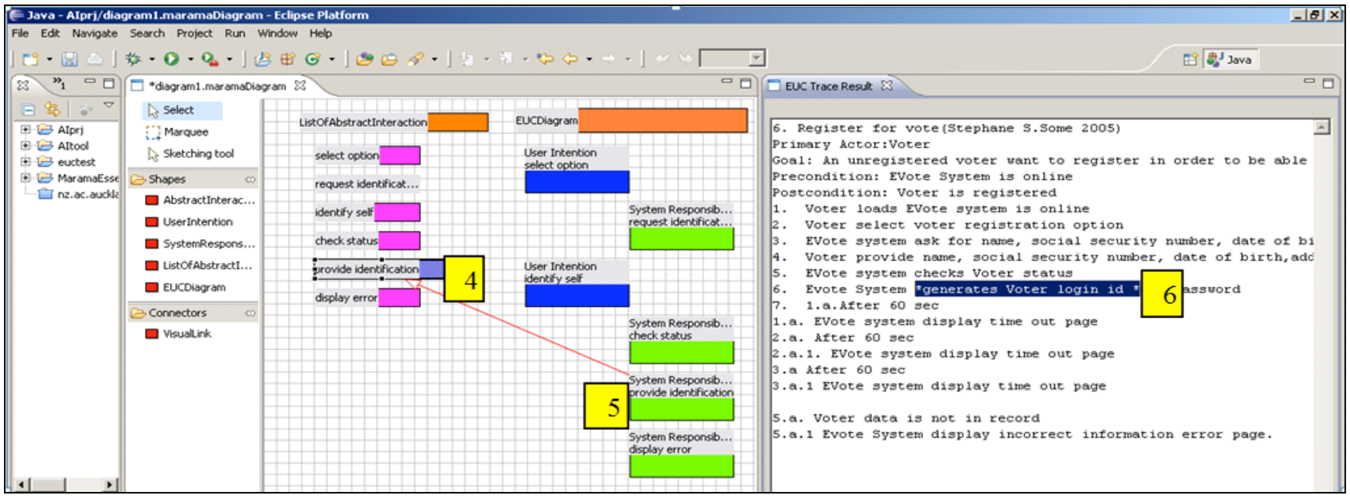
```
. Register for vote(Stephane S.Some 2005)
rimary Actor:Voter
oal: An unregistered voter want to register in orde
recondition: EVote System is online
ostcondition: Voter is registered
.   Voter loads EVote system is online
.   Voter select voter registration option
.   EVote system ask for name, social securi
.   Voter provide name, social security number, date
.   EVote system checks Voter status
.   Evote System generates Voter login id and passwo
.   1.a.After 60 sec
.a. EVote system display time out page
.a. After 60 sec
.a.1. EVote system display time out page
.a After 60 sec
.a.1 EVote system display time out page

.a. Voter data is not in record
.a.1 Evote System display incorrect information err
```

Figure 3. Tracing the abstract interaction from textual requirement and mapping to the Marama Essential

When the user selects the textual phrase the related interactions/categories are highlighted. Figure 4 shows the process of "tracing back" from the EUC diagram. The user selects a "provide identification" item (5). This highlights the related essential interaction(s) in the interactions list, in this case "provide identification" abstract interaction (4). The traceability between these items is shown by the visual link (red arrow). The corresponding textual natural language phrases are then highlighted and the matched abstract interaction will change color to purple in (4) and the matched phrases are quoted with *…* (6).The existence of these traceability links allows the consistency between these three items to be maintained. It is also possible for the tool to inform the requirements engineer if there is any item that appears to be incomplete or incorrect. The requirements engineer may modify any one of these requirements views and the tool will check the resulting models both for internal model consistency (the essential use case and EUC diagram views) and inter-model consistency (all three views). If any inconsistency occurs due to a change made by user, for example if there is a change of order, name or type for any of the abstract interaction or EUC diagram elements, an inconsistency warning will occur. If an item or phrase has been added and the new item cannot be matched to a textual requirement phrase or abstract interaction by the tracing engine an inconsistency warning will occur. If traceability relationships do not exist between phrases and items this indicates a potential incompleteness or inconsistency and no tracing result will be shown to the engineer. The tool can highlight items in one view that do not appear to be related to items in another for the engineer to investigate.

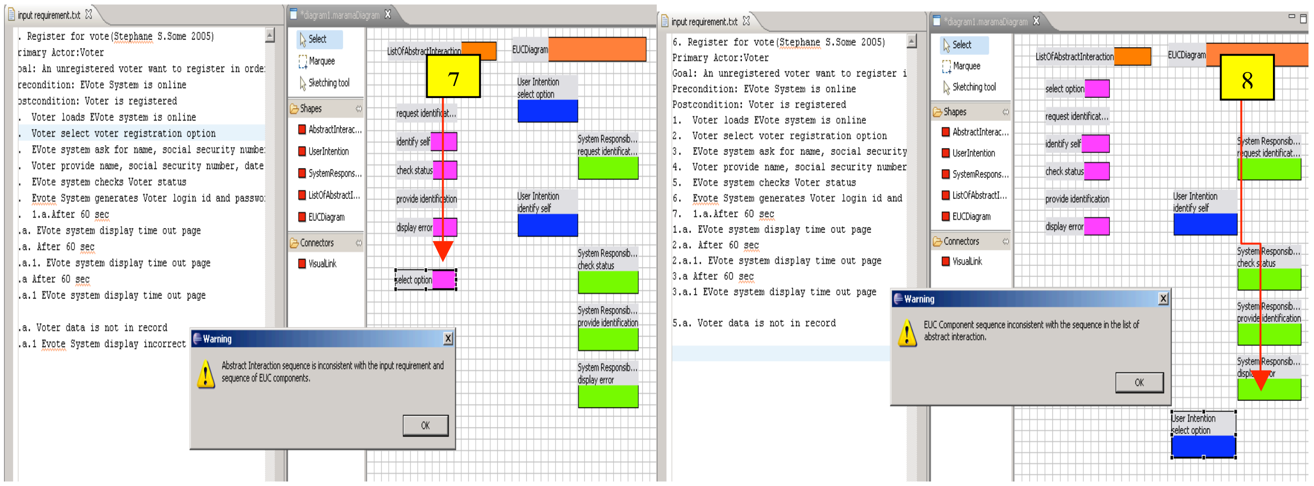
In figure 5, item (7) and (8) shows an example change of sequence to an abstract interaction - "select option". The requirements engineer has decided this should be in a different position in the set of abstract interactions. The tool has highlighted the potential inconsistency (7). This change leads also to a change of sequence and position in the EUC diagram - "select option" to the bottom. The red arrows show the change of sequence from the original position to the new one at the bottom. This produces an inconsistency in the requirements and the tool detects this and provides a warning about the inconsistency.


Java - AIprj/diagram1.maramaDiagram - Eclipse Platform
File Edit Navigate Search Project Run Window Help
AIprj
AItool
euctest
MaramaEsse
nz.ac.auckla
*diagram1.maramaDiagram
Select
Marquee
Sketching tool
Shapes
AbstractInterac…
UserIntention
SystemRespons…
ListOfAbstractI…
EUCDiagram
Connectors
VisualLink
ListOfAbstractInteraction
EUCDiagram
select option
request identificat…
identify self
check status
provide identification
display error
User Intention select option
User Intention identify self
System Responsib… request identificat…
System Responsib… check status
System Responsib… provide identification
System Responsib… display error
4
5
6
EUC Trace Result
Java

```
6. Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to register in order to be able
Precondition: EVote System is online
Postcondition: Voter is registered
1.   Voter loads EVote system is online
2.   Voter select voter registration option
3.   EVote system ask for name, social security number, date of bi
4.   Voter provide name, social security number, date of birth,adc
5.   EVote system checks Voter status
6.   Evote System *generates Voter login id *     assword
7.   1.a.After 60 sec
1.a. EVote system display time out page
2.a. After 60 sec
2.a.1. EVote system display time out page
3.a After 60 sec
3.a.1 EVote system display time out page

5.a. Voter data is not in record
5.a.1 Evote System display incorrect information error page.
```

Figure 4. Trace back from Marama Essential to the abstract interaction and textual requirement



```
. Register for vote(Stephane S.Some 2005)
rimary Actor:Voter
oal: An unregistered voter want to register in orde
recondition: EVote System is online
ostcondition: Voter is registered
.   Voter loads EVote system is online
.   Voter select voter registration option
.   EVote system ask for name, social security numbe
.   Voter provide name, social security number, date
.   EVote system checks Voter status
.   Evote System generates Voter login id and passwo
.   1.a.After 60 sec
.a. EVote system display time out page
.a. After 60 sec
.a.1. EVote system display time out page
.a After 60 sec
.a.1 EVote system display time out page

.a. Voter data is not in record
.a.1 Evote System display incorrect
```

Warning

Abstract Interaction sequence is inconsistent with the input requirement and sequence of EUC components.

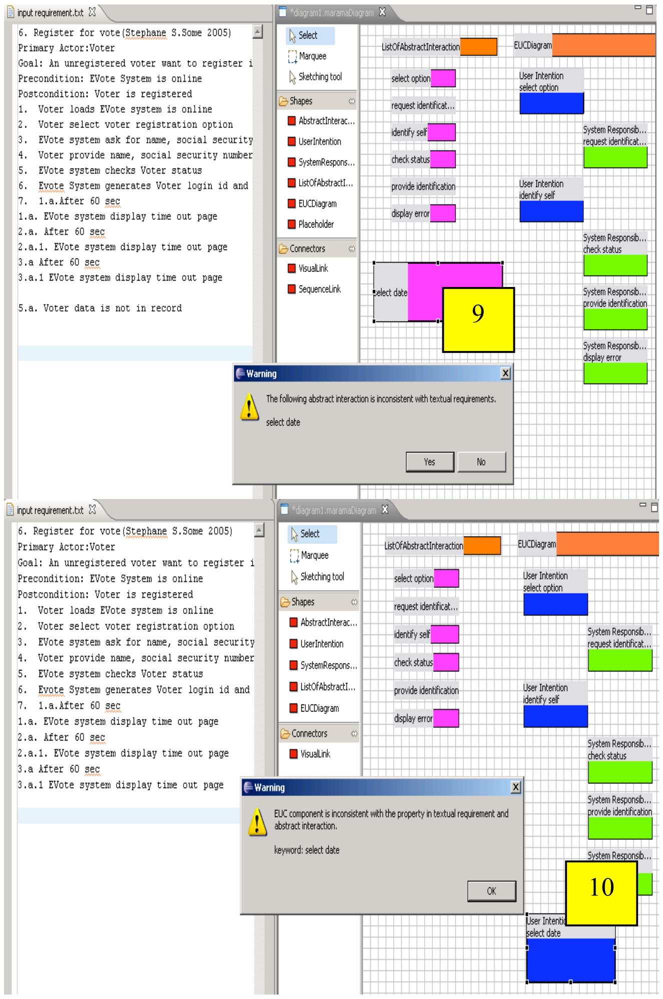
```
6. Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to register i
Precondition: EVote System is online
Postcondition: Voter is registered
1.   Voter loads EVote system is online
2.   Voter select voter registration option
3.   EVote system ask for name, social security
4.   Voter provide name, social security number
5.   EVote system checks Voter status
6.   Evote System generates Voter login id and
7.   1.a.After 60 sec
1.a. EVote system display time out page
2.a. After 60 sec
2.a.1. EVote system display time out page
3.a After 60 sec
3.a.1 EVote system display time out page

5.a. Voter data is not in record
```

Warning

EUC Component sequence inconsistent with the sequence in the list of abstract interaction.

Figure 5. Inconsistency occurring: change of the sequence of the abstract interaction and EUC diagram



```
6. Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to register i
Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter select voter registration option
3.  EVote system ask for name, social security
4.  Voter provide name, social security number
5.  EVote system checks Voter status
6.  Evote System generates Voter login id and
7.  1.a.After 60 sec
1.a. EVote system display time out page
2.a. After 60 sec
2.a.1. EVote system display time out page
3.a After 60 sec
3.a.1 EVote system display time out page

5.a. Voter data is not in record
```

Warning

The following abstract interaction is inconsistent with textual requirements.

select date

Warning

EUC component is inconsistent with the property in textual requirement and abstract interaction.

keyword: select date

Figure 6: Inconsistency occurring: adding new item to abstract interaction adding new component in EUC diagram

Figure 6. shows a potential inconsistency that happens when a new item is added to either the abstract interaction or EUC diagram views. In figure 6., item (9) and (10), a new abstract interaction has been inserted into the essential interaction view (9) and this result in a new component in the EUC diagram (10). The tool detects an inconsistency with the textual requirements and so an inconsistency warning appears and informs the requirements engineer where the inconsistency occurs. These inconsistency warnings shown in both figures illustrate the dependencies that occur between the textual requirement, abstract interaction and EUC diagram. In addition to this example usage scenario, we are also testing the tool in several other domains such as e-mobile, library system and online booking, with early positive results.

V. ARCHITECTURE AND IMPLEMENTATION

Figure 7 illustrates the architecture of Marama AI which consists of a textual requirement, abstract interaction and Marama (EUC diagram) editors. Marama AI is realized based on Marama which is built in the Java –Eclipse platform (1-2). Tools are specified using shape, meta-model and view tools and then implemented by interpretation of the specifications using a set of Marama plug-ins (4).

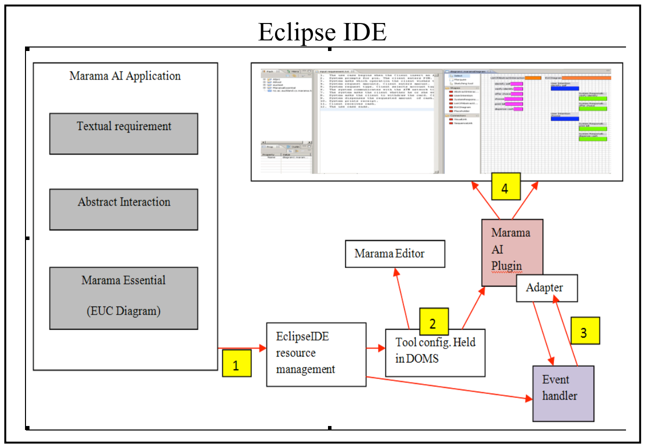


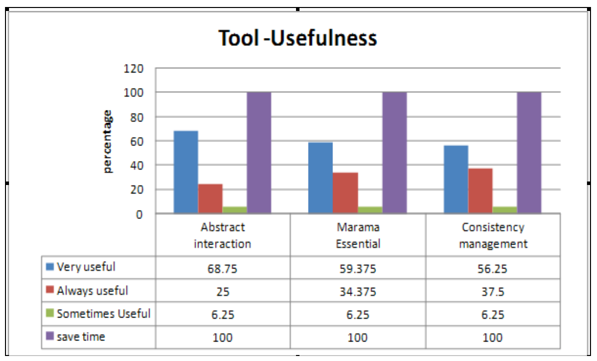
Figure 7. Marama AI Architecture

The process of extracting and mapping is assisted by event handlers (3). The event handlers help to maintain the consistency between textual requirement, abstract interaction and Marama Essential. The description of each of the event handlers is as follows. The event handler for tracing the textual requirement to the abstract interaction is called Trace. Here, the tracing engine will extract the key phrases which will be analyzed by the interaction pattern library to match with the keyword (abstract interaction). If the key phrases match with the keywords, the abstract interaction will be displayed. If there are no results displayed, it is observed that the textual requirement is normally incorrect or incomplete based on the interaction pattern. To trace back from the abstract interaction or EUC component to where it comes from, we used the help of the Trace event handler. This event handler also works together with the tracing engine. The selected abstract interaction or EUC component is analyzed by the tracing engine and then matched with the matching key phrases in the interaction pattern library. If we try to trace back the abstract interaction, the tool will show where the key phrases for that particular abstract interaction come from. If we trace back the EUC component, the system will show which abstract interaction matches with it together with the matching key phrases in the textual requirement. If no result appears, it is assumed that the requirement is either incorrect or incomplete. The requirement also is inconsistent if the users try to change the requirement by adding new abstract interactions or EUC components as shown in figure 6. The trace back event handler will not be able to trace the key phrases in the textual requirement as the new component is added without updating the textual requirement. This will also trigger the inconsistency warning to occur. The event handler for mapping the abstract interaction to Marama Essential, "Mapped to EUC", helps to generate the Essential Use Cases automatically. The event handler works with the mapping engine to map the abstract interaction to the EUC diagram. The mapping engine analyzes and matches the selected abstract interaction with the property in the interaction pattern library. Then, the abstract interaction is mapped automatically to the EUC together with it category, either user intention or system responsibility. The event handler will not map the newly added abstract interaction to the EUC component if it does not exist in the pattern library and the textual requirement is not updated in this case. This action also will trigger the inconsistency warning to inform what the inconsistency error is. The event handler Index Checker acts as a checker for the consistency of the sequence for both abstract interaction and Marama Essential. The index checker checks the index and location for each abstract interaction and EUC component. Both need to be in sequence with ordering consistent with the textual requirements. If there is any change of the sequence or location for both, the event handler provides a warning about the inconsistency that has occurred.

VI. EVALUATION

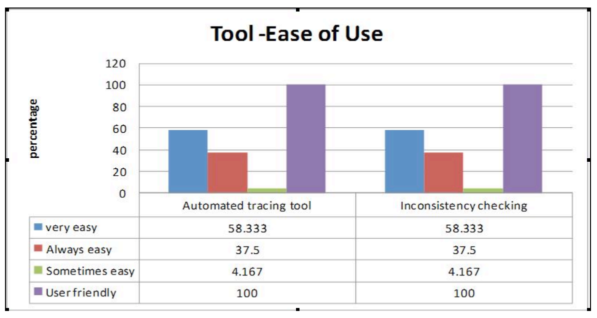
We have conducted a preliminary evaluation of the usefulness and the ease of use of Marama AI with 8 software engineering post-graduate students, several of whom had previously worked in industry as developers and/or requirements engineers. All were familiar with the EUC modeling approach. Each participant was given a brief tutorial on how to use the tool and some examples of how the EUC model is derived from the textual language requirements. The participants were asked to input their textual requirements and then use the automated tracing tool to retrieve the abstract interaction and also allow them to explore the event handler by mapping the abstract interaction to the Marama Essential, and also allow them to use the trace back facility. The participants rated the usefulness and the usability of the tool together with its inconsistency detection. They also rated the consistency between textual requirement, abstract interaction and essential use cases. Our evaluation was conducted using a standard evaluation method – a Likert scale with a five part scale was used and responses analyzed in order to determine the results shown in Figure 8 and Figure 9.

Figure 8 describes the evaluation result on the usefulness aspect of the tool. This shows that almost all of the participants agree that the automated tracing tool is useful for finding the abstract interaction, capturing requirements using the EUC model and also checking the inconsistency of the requirements. Overall the usefulness of finding abstract interactions by using our tool is almost 94%, where 69% identified it as very useful and 25% identified it as always useful. A further 6% of the participants felt that it was sometimes useful to extract the abstract interaction automatically primarily because the tool might be constrained by the domains available in the interaction pattern library. It was identified in the evaluation that approximately 94% of participants agree that using the Marama AI using the Marama Essential model is useful in capturing requirements. About 59% identified it as very useful and another 34% identified it as always useful. A further 6% of participants thought it is sometimes useful to use it as a tool in capturing requirement as they are more familiar with using UML diagrams compared to Essential Use case diagrams. For the consistency management support, approximately 94% agree that the tool provides useful inconsistency checking and maintaining the consistency of the requirements. About 56% of participants thought it very useful and around 38 % felt that the tool is always useful in managing the consistency. Another 6% of participants felt it is only sometimes useful in managing the consistency as they would like to have more complex consistency checking by the tool. All participants agree that the tool assists them in saving their time for capturing requirements and manage the consistency issue between the requirements.



	Abstract interaction	Marama Essential	Consistency management
🟦 Very useful	68.75	59.375	56.25
🟥 Always useful	25	34.375	37.5
🟩 Sometimes Useful	6.25	6.25	6.25
🟪 save time	100	100	100

Figure 8. Result of Marama AI usefulness



	Automated tracing tool	Inconsistency checking
🟦 very easy	58.333	58.333
🟥 Always easy	37.5	37.5
🟩 Sometimes easy	4.167	4.167
🟪 User friendly	100	100

Figure 9. Result of ease of use of the Marama AI

The ease of use of the automated tool was also evaluated and the results are presented in Figure 9. Both tracing and inconsistency checking features were evaluated. All of the participants agree that both components are user friendly and easy to use on the example tasks performed. For the automated tracing tool, approximate 96% agree that the tool is easy to use, where about 58% agree that the tool is very easy to use and almost 38% agree that the tool is always easy to be used. Only about 4% feel it is only sometimes easy to use. This small percentage occurs because of the difficulty they had with understanding the layout used by Marama AI. For inconsistency checking of the requirements almost 96% agree that it is easy to be handled and understand. Approximate 58% agree it is very easy to be handled and another 38% agree it is always easy to be handled. Again, only 4% of the participants thought it is sometimes easy to check the inconsistency, because the tool currently just provides warning on the detected inconsistency and no way of resolving it automatically. This minority group also wanted the tool to have an inconsistency warning together with the feedback.

VII. RELATED WORK

Many varied approaches have been proposed to maintain consistency and check the inconsistency. Olsson and Grundy developed a Web based tool to summarize the artefact data and to support basic explicit linking of element in different representational models [24]. The method uses traceability and manages fuzzy relationships between high-level software artefacts (requirement), use case model and black box test plans. The aim of this tool is to assist the inconsistency management for all changes made to artefacts. However automation is impossible and it is needed to create a relation. Besides, "*high level natural language often lack of well-defined formal abstraction for all software artefacts representation"*[18]. Cysneiro and Zisman implemented the automation generation of traceability relations among various types of models generated during the development of agent oriented systems and identification of missing elements in Promethus model and JACK code specification [18] to check completeness in order to make sure the consistency between model and code specification is maintain especially in a huge and complex system which involved different stakeholders. Rule based approaches and Promethus methodology is used with the extended version of XQuery to represents rules in traceability. Though, this is still preliminary work and completeness verification is needed for a more complete set. Another method to reduce the inconsistencies between product number is developed by [32]. XtraQue supports the generation of traceability relations in different type of documents that capable in representing different level of development lifecycle of a product line. It can define the semantics between the artefacts being compared and can also be used to bridge various activities and stakeholders taking part in the product line engineering. It generates nine traceability relations such as satisfiable, ability, dependency, overlaps, evolutions, implements, refinements, containment, similar and different feature based on OO documents created during development. The extension of XQuery is used to represent the traceability rules and consider the semantic of documents, traceability relation of various type of traceability with the product line domain and the grammatical roles of the words in textual parts of document together with the synonyms and distance of words being compared. A Rule based approach is also applied to generate automatically the traceability relations between elements of documents that are created during the development of product line system. Nevertheless, the "*existing rules failed to identify between requirements and object- oriented specification, besides changes in the documents require the traceability to be re-executed"*[32]. There is also "*method to recover traceability links between source code and free text documentation"*[33] using information retrieval which apply both the IR method namely as probabilistic and vector space. This method is applied to trace C++ and java source classes to manual pages as well as the functional requirements. However, the effectiveness of this method becoming less prominent when the number of familiar words between the source code component identifiers and the documentation item has decreases [33].

There are efforts devoted to checking inconsistency using formal and semi formal specifications. Nenwitch et al present a lightweight framework called Xlinkit in order to check consistency of distributed and heterogeneous documents using first order logic and lightweight mechanisms [7]. The main contribution of this framework is the definition of an extended semantics of first order- logic and producing the hyperlinks which diagnose well the inconsistency across the specification compared to the Boolean result. The incremental checking technique used is also able to decrease the checking time. However, XLinkit limitation is a lack of discovery of problems if the inconsistencies are recognized no action is taken in if the inconsistency problem becomes complex.

Egyed implemented a UML-based transformation framework to check the inconsistency and help in comparison. The author introduced an automated checking tool called as VIEWINTEGRA which used the consistent transformation to translate diagrams into interpretations and used the consistency comparison to compare those interpretations to other diagrams [4]. This method can check inconsistencies without the help of third party or intermediate languages. The limitation of this tool exist when checking the consistency between object diagram and state chart diagram or vice versa, as they couldn't be transformed directly and need to be changed to a class diagram first in order to obtain the consistency results [4].

All the literature referenced mentions the use of traceability, formal and semi formal specification in managing the consistency between different representation models, code level and documents. Almost none of the works stressed the use of traceability in managing consistency between the textual requirement and model representation especially between textual requirements and an Essential Use Case model. The traceability elaborated also does not apply any visualization approach or tool to visualize the traceability and the consistency between these components.

VIII. SUMMARY

We have discussed the advantages of using a traceability approach in managing consistency between textual requirements, abstract interactions and Essential Use Cases (EUCs). Traceability and consistency between these artefacts are visualized with the support of Marama. We described a proof of concept support environment, Marama AI that generates tracing and mapping between textual requirements, abstract interactions and EUCs. This tool also assists users and requirements engineers in capturing requirements and generates EUCs automatically. MaramaAI is able to minimize human intervention in checking consistency. A preliminary evaluation was conducted showing promising results for usefulness and ease of use. Key future work involves extending our tool to check higher level consistency between textual requirements, abstract interactions and EUCs. We will use Essential Use Case patterns to compare the generated EUC models against to try and detect where they diverge from accepted patterns of use. This will allow us to identify further examples of incompleteness and inconsistency in their originating textual. Visualisation in Marama essential will be further improved and the interaction pattern library will be further enhanced to support wider domains. Further evaluation in the aspect of efficacy and performance of the tool will also be done with a larger group of participants. We would like to assess not only the impact of our tool both in terms of improving the adoption and use of the Essential Use Case method, but also its impact on improving the efficacy of the method itself. This may include integration with other requirements and design modelling views.

Acknowledgment

Kamalrudin is funded by the Malaysia ministry of Higher Education (MOHE) and the University of Auckland.

References

[1] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," Information and Software Technology, vol. 45, 2003, pp. 993-1009.

[2] C. Denger, 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 Computer Society, 2003, pp. 80

[3] A. Satyajit, M. Hrushikesha, and C. George, "Domain consistency in requirements specification," in Quality Software, 2005. (QSIC 2005). Fifth International Conference on, 2005, pp. 231-238.

[4] A. Egyed, "Scalable Consistency Checking Between Diagrams-The ViewIntegra Approach," in Proceedings of the 16th IEEE international conference on Automated software engineering: IEEE Computer Society, 2001, p. 387.

[5] A. Kozlenkov and A. Zisman, "Are their design specifications consistent with our requirements?," in Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on, 2002, pp. 145-154.

[6] A. Egyed, "Instant consistency checking for the UML," in Proceedings of the 28th international conference on Software engineering Shanghai, China: ACM, 2006, pp. 381-390.

[7] C. Nentwich, W. Emmerich, A. Finkelstein, and E. Ellmer, "Flexible consistency checking," ACM Trans. Softw. Eng. Methodol., vol. 12, pp. 28-63, 2003.

[8] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," ACM Trans. Softw. Eng. Methodol., vol. 14, pp. 277-330, 2005.

[9] L. G. Gnesi S, Trentanni G, Fabbrini F, Fusani M, "An automatic tool for the analysis of natural language requirements," International Journal of Computer Systems Science & Engineering, vol. 20, pp. 53-61, 2005.

[10] J. Bowen and V. Stavridou, "Safety-critical systems, formal methods and standards," Software Engineering Journal, vol. 8, pp. 189-209, 1993.

[11] K. Haruhiko and S. Motoshi, "Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach," in Proceedings of the Fifth International Conference on Quality Software: IEEE Computer Society, 2005.

[12] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating Natural Language specifications from UML class diagrams," Requirements Engineering, vol. 13, pp. 1-18, 2008.

[13] M. Kamalrudin, "Automated Software Tool Support for Checking the Inconsistency of Requirements," in 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, 2009.

[14] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," ACM Trans. Softw. Eng. Methodol., vol. 16, p. 13, 2007.

[15] I. S. Gerald Kotonya, Requirement Engineering Process and Techniques. West Sussex,England: John Wiley & Sons Ltd, 1998.

[16] O.M. ÄÄLINOJA Juho "Software requirements implementation and management," Software & systems engineering and their applications vol. vol. vol.1 à 3, , pp. pp. 1.1-1.8Note(s) 2004

[17] M. F. Bashir and M. A. Qadir, "Traceability Techniques: A Critical Study," in Multitopic Conference, 2006. INMIC '06. IEEE, 2006, pp. 265-268.

[18] G. Cysneiros and A. Zisman, "Traceability and completeness checking for agent-oriented systems," in Proceedings of the 2008 ACM symposium on Applied computing Fortaleza, Ceara, Brazil: ACM, 2008, pp. 71-77.

[19] W. L. Poon and A. Finkelstein, "Consistency management for multiple perspective software development," in Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops San Francisco, California, United States: ACM, 1996, pp. 192-196.

[20] A. Finkelstein, "A Foolish Consistency: Technical Challenges in Consistency Management," in Database and Expert Systems Applications, 2000, pp. 1-5.

[21] N. Christian, E. Wolfgang, and F. Anthony, "Consistency management with repair actions," in Proceedings of the 25th International Conference on Software Engineering Portland, Oregon: IEEE Computer Society, 2003.

[22] B. Nuseibeh, S. Easterbrook, and A. Russo, "Leveraging Inconsistency in Software Development," Computer, vol. 33, pp. 24-29, 2000.

[23] J. Grundy, J. Hosking, and W. B. Mugridge, "Inconsistency management for multiple-view software development environments," Software Engineering, IEEE Transactions on, vol. 24, pp. 960-981, 1998.

[24] T.a. Olson, J.Grundy, "Supporting Traceability and Inconsistency Management Between Software Artefacts," in Proceedings of the IASTED International Conference on Software Engineering and Applications, ,Boston, MA, November 2002.

[25] L. L. Constantine, "Essential modeling: use cases for user interfaces," interactions, vol. 2, pp. 34-46, 1995.

[26] R. Biddle, J. Noble, and E. Tempero, "Essential use cases and responsibility in object-oriented development," Aust. Comput. Sci. Commun., vol. 24, pp. 7-16, 2002.

[27] L. L. Constantine and A. D. L. Lockwood, "Structure and style in use cases for user interface design," in Object modeling and user interface design: designing interactive systems: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 245-279.

[28] A. D. L. Lockwood. and L. L. Constantine, Software For Use: A Pactical Guide to the Models and Methods of Usage- Centered Design: Addison Wesley Longman,Inc, 1999.

[29] J. C. Grundy, J.G Hosking, J.Huh, and N.Li, "Marama: an Eclipse meta-toolset for generating multi-view environments," in 2008 IEEE/ACM International Conference on Software Engineering, Liepzig, Germany, May 2008,.

[30] J. Noble. R.Biddle, E. Tempero, "Pattern for Essential Use Cases," Victoria University of Wellington, Wellington,New Zealand April 2000.

[31] S. S. Some, "Use cases based requirements validation with scenarios," in Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on, 2005, pp. 465-466.

[32] W. J. a. A. Zisman, "XTraQue: traceability for product line systems," Software and Systems Modeling, September 05, 2007.

[33] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," Software Engineering, IEEE Transactions on, vol. 28, pp. 970-983, 2002.