MaramaAIC: Tool Support for Consistency Management and Validation of Requirements

[1]Massila Kamalrudin, [2]John Hosking, [3]John Grundy

[1]*Innovative Software System & Services Group,*
Universiti Teknikal Malaysia Melaka, Melaka, Malaysia.
massila@utem.edu.my
[2]*Faculty of Science, University of Auckland, Auckland, New Zealand*
j.hosking@auckland.ac.nz
[3]*School of Software and Electrical Engineering, Swinburne, University of Technology, PO Box 218, Hawthorn, Victoria 3122, Australia*
jgrundy@swin.edu.au

Abstract: Requirements captured by requirements engineers are commonly inconsistent with their client's intended requirements and are often error prone. There is limited tool support providing end-to-end support between the requirements engineers and their client for the validation and improvement of these requirements. We have developed an automated tool called MaramaAIC (Automated Inconsistency Checker) to address these problems. MaramaAIC provides automated requirements traceability and visual support to identify and highlight inconsistency, incorrectness and incompleteness in captured requirements. MaramaAIC provides an end-to-end rapid prototyping approach together with a patterns library that helps to capture requirements and check the consistency of requirements that have been expressed in textual natural language requirements and then extracted to semi-formal abstract interactions, Essential Use Cases and User Interface prototype models. It helps engineers to validate the correctness and completeness of the Essential Use Case modelled requirements by comparing them to "best-practice" templates and generates as abstract prototype in the form of Essential User Interface (EUI) prototype models and concrete User Interface (UI) views in the form of HTML. We describe its design and implementation together with results of evaluating our tool's efficacy and performance, and user perception of the tool's usability and its strengths and weaknesses via a substantial usability study. We also present a qualitative study on the effectiveness of the tool's end-to-end rapid prototyping approach in improving dialogue between the Requirements Engineer and the client as well as improving the quality of the requirements.

Keyword: consistency management, requirements validation

Introduction

A set of requirements is interpreted at the early phase of a system development (Kotonya 1998) and it reflects the client's need for a system. It describes "*how the system should behave, constraints on the system's application domain information, constraints on the system operation or specification of a system property or*

attribute" (Kotonya 1998). Software requirement specifications elaborate the functional and non-functional requirements, design artifacts, business processes and other aspects of a software system. Software requirement specifications that are complete and accepted by developers and clients provide a shared understanding and agreement of what a software system should do and why. Since requirement documents form the basis of this agreement and subsequent development processes, they should be correct, complete, and unambiguous (Denger, Berry & Kamsties 2003) and need to be analysed with respect to Consistency, Completeness and Correctness (the "3 Cs") to detect errors such as inconsistency and incompleteness (Biddle, Noble & Tempero 2002).

It is common to find inconsistencies in requirements specifications as the requirements elicitation process involves two or more parties in delivering and understanding correct requirements. Zowghi et al.(2003) assert that expression by different stakeholders may lead to inconsistencies and contradictions because the parties keep changing their minds throughout the development process. *Inconsistent requirements* occur when two or more stakeholders have differing, conflicting requirements and/or the captured requirements from stakeholders are internally inconsistent when two or more elements overlap and are not aligned (Zisman 2001), (Nuseibeh, Easterbrook & Russo 2000). Typically the relationship is articulated as a consistency rule against which a description can be checked. Inconsistency in requirements also occurs when there are incorrect actions (Fabbrini, Fusani, Gnesi & Lami 2001), or where requirements clash because of disagreements about opinions and bad dependencies (Satyajit, Hrushikesha, & George 2005), sometimes resulting from a lack of skill of different users dealing with shared or related objects. In addition, Litvak (2003) believes that inconsistency occurs when the same parts of the model are portrayed by multiple diagrams and Lamsweerde et al. (1998) find that inconsistency occurs in a set of descriptions when the descriptions can't be satisfied together.

In the context of our research, inconsistencies happen when any of the requirements components that are intended to be equivalent are not. This could be by not being in the same sequence, not having the same name, not being consistent when equivalent components are changed and not being consistent across differing representational models. Positive and negative outcomes for the system development lifecycle are caused by inconsistency (Zisman 2001). Inconsistency

highlights contradictory views, perceptions and goals among stakeholders who are involved in a particular development process. It also helps to identify which parts of the system needs further analysis, as well as helping to facilitate the discovery and evocation of the options and information of a system. In addition, Nuseibeh et al. believe that inconsistency can be used as a tool to verify and validate the software process (Nuseibeh et al. 2000). However, it is still vital to avoid or check for inconsistency as it could affect the whole development process, as the clients' requirement needs cannot be met and attempts to do so may cause delay, increase the cost of the system development process, put at risk properties related to the quality of a system and make the maintenance process of a system cumbersome.

Use cases have been used in Requirements Engineering for many years as a semi-formal way to model software requirements from the perspective of user interaction with a system (Jacobson et al. 1999). They are one of the most widely used approaches to semi-fomal requirements modelling and a number of research efforts have focused on their capture and analysis. Essential Use Cases, developed by Constantine and Lockwood, offer a simplified, more abstract approach to modelling user/system interaction. They offer advantages of a more abstract and thus simpler form, more structured rules concerning the capture of the EUC models from natural language requirements, and a set of best-practice EUC patterns than can be used for analysis of incorrectness, inconsistency and incompleteness issues (constantine and Lockwood, 1999). Rapid user interface prototypes have been used, often with User Case scenarios, to help refine use case-based requirements by modelling simplified forms of user interfaces to target systems. They offer a way to enhance use case-based requirements especially for stakeholders where a model of the target system interface can be visualized early during requirements capture to give an idea of what it will look like and potentially behave. Essential User Interface prototypes can be used with EUC models to again provide more abstract representations of key target interface components to assist in requirements capture, and for incompleteness and incorrectness analysis.

In order to help engineers to achieve consistency of requirements and to help control and track requirement changes, good tool support is needed (Yufei, Tao, Tianhua & Lin 2010). There are many commercial requirements management tools including DOORS (Hull, Jackson & Dick 2005), Serena RTM (Inc. 2011), Caliber RM (Corporation 1994 - 2010) and Requisite Pro (IBM). They provide good

coverage of requirements management but limited analysis and validation support such as for consistency, correctness and completeness (Geisser 2007). Many research tools exist, most tending to focus on a partial solution for a particular requirements management process, formalism or analysis task. Use case supporting tools are very common, including many commercial tools but also a range of research prototypes that focus on capture from natural language, translation of UC models into other formal models, and UC-based consistency checking. Essential Use Case modelling tools are limited and have very limited consistency checking support. A range of rapid user interface prototyping tools exist for the requirements engineering domain, but few are integrated with UC or EUC models, and few focus on requirements elicitation and consistency enhancement per se.

Given this need for imporved requirements capture and analysis, and current lack of adequate tool support, we wanted to address these issues by helping requirements engieers to capture elicited requirements in a semi-formal manner using the Esential Use Case approach. We wanted to leverage this semi-formal model to provide 3Cs checking for the captured EUC-based requirements models. To achieve this we developed novel automated tool support - called MaramaAIC - that uses semi-formal Essential Use Case (EUC) models and Essential User Interface (EUI) prototype models to support consistency management and requirements validation. Both modelling approaches were chosen as they work well in tandem and focus on presenting the abstract, essential requirements models focusing on user/system interaction (Constantine & Lockwood 2003). Our research question derived from this approach is thus "***can automated support for Essential Use Case and Essential User Interface modelling enhance the consistency management and validation of requirements over manual methods?".*** We have developed a prototype tool and compared its performance to manual extraction and validation methods to try and answer this research question.

The remainder of this paper is organized as follows. Section 2 explains the background of the study by defining the basic terms that are used in this paper and Section 3 discusses related work. The automated tool support is discussed in Section 4 and the pattern libraries are discussed in Section 5. Section 6 illustrates an example of the tool's usage and Section 7 discusses the architecture and the implementation of the tool. The results of the evaluation are discussed in section 8 and the paper concludes with a summary and future work options.

Background

Use Cases (UC)

Use case modelling of requirements has been used for many years in research and practice. The UML popularized their usage for many software development projects (Jacobson et al. 1999). The key concept of the Use Case is a set of related interactions between a target system end user and a part of the system to achieve a task. A use case describes user interacitons with the system as a flow of interactions and responses, and may include pre- and post-conditions (when the use case may be used ; pre-existing system or other state information ; and changes to the state post the use case completion); alternative or exception flows; information required by steps in the use case that is exchanged between the user and system; and sometimes other annotations to indicate system behavior in response to interactions. Scenarios are often used with Use Cases to capture particular examples of user/system interaction including example information exchanged. A great advantage of UC-based models is simplicity of concept, understandability by a wide range of stakeholders, usefulness in constructing acceptance and other tests, and semi-formal nature. Disadvantages are lack of a formalism resulting in limited ability to check for 3Cs problems, lack of agreement on semantics, informality of meaning due to use of natural langage and domain-specific terminology, and the large number of use cases that are needed to model even moderately complex systems.

Common alternatives are more formal representations of requirements, such as i* (Yu, 1997), KAOS (Dardenne et al. 1993), various logics such as TLA and LTL (Lamport, 2002), and visualisations of information structure and flow, such as activity diagrams and sequence diagrams (Jacobson et al. 1999).. Advantages of more formal approaches include their ability to be checked by theorem provers and model checks. Advantages of more diagrammatic forms include ability to model larger aspects of systems, more clearly show alternative and other flows, and ability to show related artefacts impacted by user/system interactions. Disadvantages compared to use case modelling include challenges in scaling the models, particularly many formal models, need for mathematical or other logical understanding by model users, and need for good diagrammatic model authoring tools.

Essential Use Cases (EUC)

The EUC approach is defined by its creators, Constantine and Lockwood, 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*" (Constantine & Lockwood 1999). An EUC takes the form of a dialogue between the user and the system. The aim is to support better communication between developers and stakeholders via a technology-free model and to assist better requirements capture. This is achieved by allowing only specific detail relevant to the intended design to be captured (Biddle, Noble & Tempero 2002). Compared to a conventional UML use case, an equivalent EUC description is generally shorter and simpler as it only comprises the essential steps (core requirements) of intrinsic user interest (Biddle et al. 2002). It contains user intentions and system responsibilities to document the user/system interaction without the need to describe a user interface in detail. The abstractions used are more focused towards the steps of the use case rather than narrating the use case as a whole.

A set of essential interactions between user and system are organised into an interaction sequence. Consequently, an EUC specifies the sequence of the abstract steps and captures the core part of the requirements (Biddle et al. 2002). Furthermore, the concept of responsibility in EUC aims to identify "*what the system must do to support the use case" without being concerned about "how it should be done*" (Biddle et al. 2002). By exploiting the EUC concept of responsibility, a fruitful research area is to focus on the consistency issues between responsibility concepts in requirements and their related designs. This can potentially be used to improve traceability support. EUCs also benefit the development process as they fit a "*problem-oriented rather than solution–oriented*" approach and thus potentially allow the designers and implementers of the user interface to explore more possibilities (Blackwell et al. 2001) They also allow more rapid development: by using EUCs, it is not necessary to design an actual user interface (Biddle et al. 2002).

Figure 1 shows an example of a textual natural language requirement (left hand side) and an example Essential Use Case (right hand side) capturing this

requirement (adapted from (Constantine & Lockwood 2001)). On the left is the textual natural language requirement from which important phrases are extracted (highlighted). From each of these, a specific key phrase (essential requirement) called an abstract interaction is abstracted and is shown in the Essential Use case on the right as user intentions and system responsibilities.

This assists in abstracting the requirements away from for specific technologies. For example, the requirement of typing in login information compared to using biometrics as alternative identification technologies are transformed to a more abstract expression of requirement called "identify self".

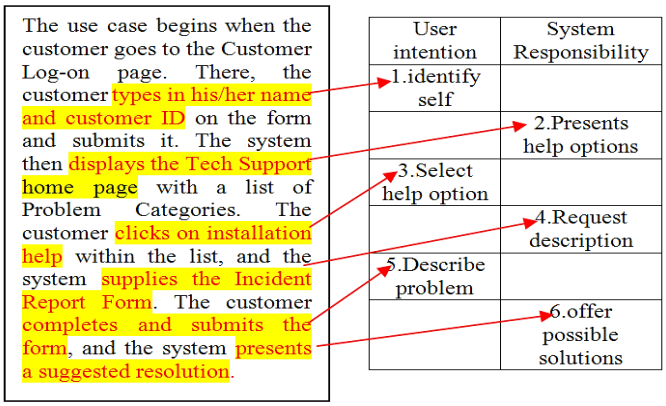


Fig 1. Example of generated EUC model (right) from the textual natural language requirements (left) adapted from (Constantine& Lockwood 2001).

Although EUCs simplify captured requirements compared to conventional UML use cases, requirements engineers still face the problem of correctly defining the level of abstraction which requires effort and time. (Biddle et al., 2000). Requirements engineers need to abstract the essential requirements (using the EUC concept of abstract interactions) manually. This involves understanding the natural language requirements and then extracting an appropriately abstract essential requirement embedded in a logical interaction sequence.

Rapid User Interface Prototyping

Rapid prototyping assists the requirement elicitation process by supporting requirements engineers to gain early feedback from clients on the captured requirements by putting them into a more tangible form i.e. a model of the target system user interface implementing those requirements (Robertson 2006), (Buskirk & Moroney 2003). Low-fidelity or abstract prototypes (often paper) are commonly

used in this process (Constantine 1998). The idea is to make the captured requirements for e.g. a use case much more tangible to the stakeholder by giving them a semblance of the target system if it were implemented based on these requirements. The stakeholder is then able to more concretely understand how the requirements might be realized and if the captured requirements are indeed consistent, complete and correct. A rapid User Interface (UI) prototyping approach can thus complement other capture and checking approaches used.

Types of abstract prototypes include abstract user interfaces (Cristian 2008), UI prototypes (Memmel & Reiterer 2009) and EUI prototypes (Constantine & Lockwood 1999). These are all easy-to-change mock ups which encourage iteration of the elicitation and validation process (Robertson 2006), (Memmel & Reiterer 2009). They allow a rough walk-through of user tasks before needing to factor in hardware or technology concerns (Buskirk & Moroney 2003) and can avoid clients being fixated at an early stage on concrete product appearance rather than functionality (Robertson 2006). However, previous work has shown that the application of low-fidelity techniques in practice can prove challenging (Robertson 2006), due to lack of tool support and lack of integration between models, processes and analysis support.

An example of rapid prototyping is Essential User Interface (EUI) prototyping, a low-fidelity prototyping approach (Ambler 2003-2009). It provides the general idea behind the UI but not its exact details. It focuses on the requirements and not the design, representing UI requirements without the need for prototyping tools or widgets to draw the UI (Constantine & Lockwood 2003). EUI prototyping extends from, and works in tandem with, the semi-formal representation of EUCs, both focusing on users and their usage of the system, rather than on system features (Ambler 2004). It thus helps to avoid clients and REs being misled or confused by chaotic, rapidly evolving and distracting details. Being primarily a whiteboard or paper-based technique to date, it does not integrate well with most other tools used in the software engineering process (Ambler 2003-2009). However, it shows promise as a way to complement EUC-based semi-formal models by surfacing the requirements using an abstract user interface model.

Figure 2, from Ambler (2004), shows an example of an EUI prototype being developed from an Essential Use Case (EUC). The post-it notes represent abstractions of user interfaces. The different colours of these notes represent

different UI elements. Pink notes represent the input field, yellow notes represent display only and blue notes represent actions (Ambler 2004). Here, the Requirements Engineer (RE) is capturing the user intention/system responsibility dialogue represented in the EUC as possible UI functionality at a high level of abstraction.

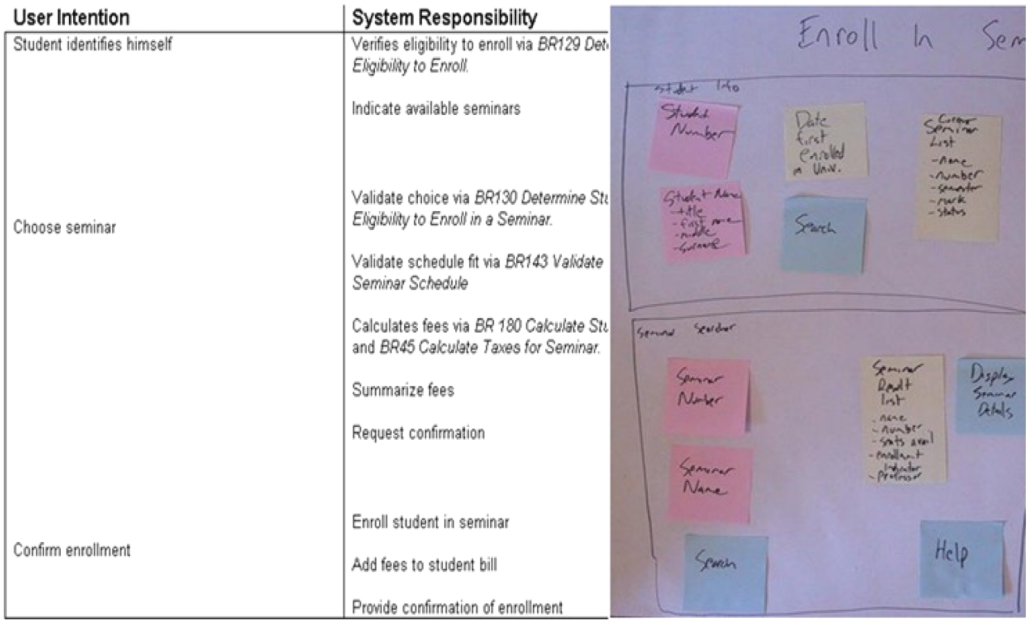


Fig 2. Example of EUI prototype iterates from Essential Use Cases ((Ambler 2004)), (Hull, Jackson, & Dick 2005))

Related Work

Much research has been devoted to developing tools for managing the consistency of or checking for inconsistency in requirements using formal or semi-formal specifications. For example, E-Lopez-Herrejon and Egyed (2012) presented work on fixing inconsistencies with variability by using and comparing two approaches: random and heuristic, based on the size of the fixing set and the time taken for the execution by having a DFS-based approach as a baseline. However, further consideration is needed for both fixing multiple inconsistency rule instances and having more complex fixing operations (Lopez-Herrejon & Egyed 2012). Other work by Reder and Egyed (2012) provides an automated tool for incrementally validating design rules in a validation tree to improve the performance of incremental consistency checking. Their automated tool support was found to be able to minimise the time taken for re-validation of design rule and fits well with

all kind of design rules. However, it is only focussed on validating parts that are affected by model changes and not all design rules (Reder & Egyed 2012). Egyed (2001) has also implemented a UML-based transformation framework to check inconsistency and help in comparison using an automated checking tool called VIEWINTEGRA. This uses consistent transformation to translate diagrams into interpretations and used the consistency comparison to compare those interpretations with those of other diagrams (Egyed 2001). This technique can check inconsistencies without the help of third party or intermediate languages. The limitation of this tool exists when checking the consistency between an object diagram and state chart diagram or vice versa, as they cannot be transformed directly and need to be changed to a class diagram first in order to obtain consistency results (Egyed 2001).

Another approach is presented by Perrouin et al. (2009) for managing the inconsistencies amongst heterogeneous models by using a model composition mechanism. The information from the heterogeneous models is translated to a set of model fragments (Perrouin, Brottier, Baudry & Le Traon 2009). Fusion is applied to build a global model which allows various inconsistencies to be detected, resulting in the global model (Perrouin, Brottier, Baudry & Le Traon 2009). Automation is applied to compute traceability links between the input model and the global one and thus support the reporting of the inconsistencies on the original model and help to resolve the cause of those inconsistencies Perrouin, Brottier, Baudry & Le Traon 2009). However, a classification of which inconsistencies need to be resolved is not provided (Perrouin, Brottier, Baudry & Le Traon).

Nentwich et al. (2003) proposed a repair framework for inconsistent, distributed documents (Nentwich, Wolfgang & Anthony, 2003). They generate interactive repairs from a first order logic formula that constrains the documents. Their repair system provides a correct repair action for each inconsistency together with available choices. However, they face problems when the repair actions interact with the grammar in a document, and also actions generated by other constraints (Nentwich,Emmerich, Finkelstein and Ellmer 2003). Their approach also fails to identify a single inconsistency that may lead to other inconsistencies (Nentwich, Wolfgang & Anthony 2003). Gervasi and Zowghi (2005) used the tool in detecting, analysing and handling inconsistencies in requirements for various stakeholders. This work extended the tool to employ theorem proving and model

checking in the context of default logic to deal with the problems in a formal manner. The tool's limitation is that propositional logic used is not powerful enough to model complex system behaviour (Gervasi & Zowghi 2005).

There has also been work done to check the consistency of aspect-oriented requirements. Sardinha et al. (2012) developed an automated conflict detector called Early Aspect Analyzer (EA-Analyzer) based on a Bayesian learning method for a large set of aspect-oriented requirements compositions. This tool demonstrates the benefits of Aspect Oriented Requirements Engineering (AORE) to detect and to analyse conflicts in the AO requirements text, but it requires training before using the tool and needs wider requirements sets to test the scalability of the tool. The tool also does not operate alone as it requires assistance from the EA-Miner tool which is developed by Sampaio et al. (2005) to identify and separate concerns, either aspectual or non-aspectual (Sampaio et al. 2005). Other work relating to checking consistency using an aspect-oriented paradigm, this time for web applications, is by Yijun (2004). The author presents a tool called HILA which was designed as an extension of UML state machines to model the adaptation rules for web applications (Yijun 2004). However, this work is not limited to web engineering applications but may also be applicable to other areas (Yijun 2004). HILA could be helpful in improving the modularity of models and helps to automate the consistency checking of aspects to ensure rules are always in a consistent state (Yijun 2004). Likewise, Zhang and Holzl (2012) uses HILA with their weaving altgorithm and implementation of semantic aspects to check and to resolve conflicts between various aspects. Then Zhang (2012) also uses HILA to model the mutual exclusion requirements in a specified place. This work is also found could minimises potential conflicts between aspects. On the other hand, Yue et al. (2015) developed a method and tool called aToucan to automatically transform use case model to analysis models such as class, sequence and activity diagrams. Here, the traceability is established between both the use case and analysis model where it help to maintain the models when changes happen and somehow indirectly help to ensure complete, correct and consistent UML model comprising of both structural and behavioral aspects via an intermediate model to be generated.

Nguyen et al. (2012) developed an automated tool called REInDetector a knowledge-based engineering tool to capture and to detect a range of

inconsistencies of requirements. This tool uses descriptic logic (DL) as its formal basis of object/class-style ontologies to formalise and analyse requirements (Nguyen, Vo, Lumpe & Grundy 2012). The tool can identify missing elements and conflicts in requirements (Nguyen et al. 2012). However, there is very limited support for temporal operators in DL and this does not allow the tool to detect conflicts associated with the requirements that are not expressible in the DL.

To summarize, many techniques discussed above are reasonably well developed and evaluated but most are immature. Most work uses tool support for the checking process. However, most of these integrate with other available tools and are not purely built for consistency checking, especially when this needs to deal with processing natural language or to formalize the requirements. Most tools or approaches lack rigorous checking for consistency as they only support partial solutions for checking or identifying inconsistency and with a homogeneous model of a set of requirements. We also identified that the tools developed need human intervention to interpret the consistency results or invoke actions to check for inconsistency. Semi-formal specifications are of great interest although some studies concluded that maintaining consistency between models is not important and expensive (Kovacevic 1999). Almost no research has been undertaken on managing consistency using the Essential Use Case representation (Biddle April 2000). Very little of the identified research work provides tools to handle full end-to-end consistency checking support, i.e from the natural language requirement to models and then to a user interface prototype. Most work is only concerned with validating requirements by requirements engineers and not by the clients.

Preliminary Experience in Applying EUCs and EUIs

Previous research using the EUC approach to model software requirements has indicated that requirements engineers sometimes have difficulty in identifying the "abstract interactions" used by EUCs and their sequencing (Biddle.R April 2000). To obtain a better understanding of these potential difficulties, we conducted a user study with 11 post-graduate software engineering students, several of whom had previously worked in industry as developers and/or requirements engineers. All were very familiar with UML use case modeling and most had used UML use cases to model requirements previously. None were familiar with the EUC modeling approach. This allowed us to see ways in which novice EUC users could be

supported by a tool. Though we used students and inexperienced requirements engineers without much EUC experience, it does not impact our study results as we wanted to precisely understand the challenges faced by such novice EUC users.

The participants carried out the extraction of an EUC model from a set of requirements specified in natural language, in order to observe their performance and understand their experiences in using EUCs. We used the same sets of requirements for modelling (Constantine & Lockwood 2001) and compared the EUC models developed by EUC novices with the ones produced by a modeler familiar with EUCs (Kamalrudin 2010) (Kamalrudin & Grundy 2011).

In this study the average time taken to accomplish the EUC development task was 11.2 minutes. The longest time taken was about 25 minutes and the shortest time taken was about 5 minutes, so there was significant variation in the time taken. Also participants were more likely to generate incorrect EUC interactions than correct ones, and very unlikely (9.1%) to produce a completely correct EUC. All but one participant failed to identify some of the essential interactions present in the natural language requirements; many failed to assemble these into an appropriate interaction sequence. The root cause of most problems was that participants tended to incorrectly determine the required level of abstraction for their essential interactions (the user intentions and system responsibilities of the EUC model). This is based on observations made as they performed the task as well as analysis of the answers provided by them. The study also demonstrates that it was quite time consuming for participants as they needed to figure out the appropriate keywords that describe each abstract interaction and to organise them into an appropriate sequence of user intentions and system responsibilities.

We then conducted a similar study with the same scenario to understand further the problem faced by requirements engineer in applying the EUI prototype model approach. This second study involved 20 post-graduate software engineering students, several of whom had previously worked in the industry as developers and/or Requirements engineers. All were familiar with requirements and prototyping at the elicitation phase, but none with the EUI prototyping approach. Each participant was given a brief tutorial on the approach and examples of natural language requirements with derived EUC models and EUI prototypes. Participants were then asked to develop an EUI prototype model from an EUC model and natural

language requirements. Here, we also tracked the time they took to complete their tasks.

As with the EUC study, participants were found to be more likely to generate incorrect EUI prototype models than correct ones. This is because the participants tended to incorrectly determine the main UI component of a specific business use case. Almost all participants tended to capture unnecessary UI components, gearing towards a concrete GUI rather than EUI components. There was also considerable variation in the time taken and the longest time taken did not increase the likelihood of the correctness of the answer. Our studies thus support the anecdotal findings reported in (Biddle 2000) regarding the problems faced in extracting the correct abstract interaction of EUCs and using low-fidelity prototypes but with more quantitative evidence.

Automated Tool Support: MaramaAIC

The results of these preliminary studies motivated us to develop new automated tool support to enable requirements engineers to effectively capture or confirm more requirements with clients at an early stage of requirement analysis. We wanted to support an end-to-end rapid prototyping approach which uses low-fidelity EUI prototyping together with a concrete UI prototype. Our new tool, MaramaAIC (Automated Inconsistency Checker) provides a range of inconsistency checking which is not limited to a partial solution or partial components to be checked. Figure 3 shows the way MaramaAIC is used.

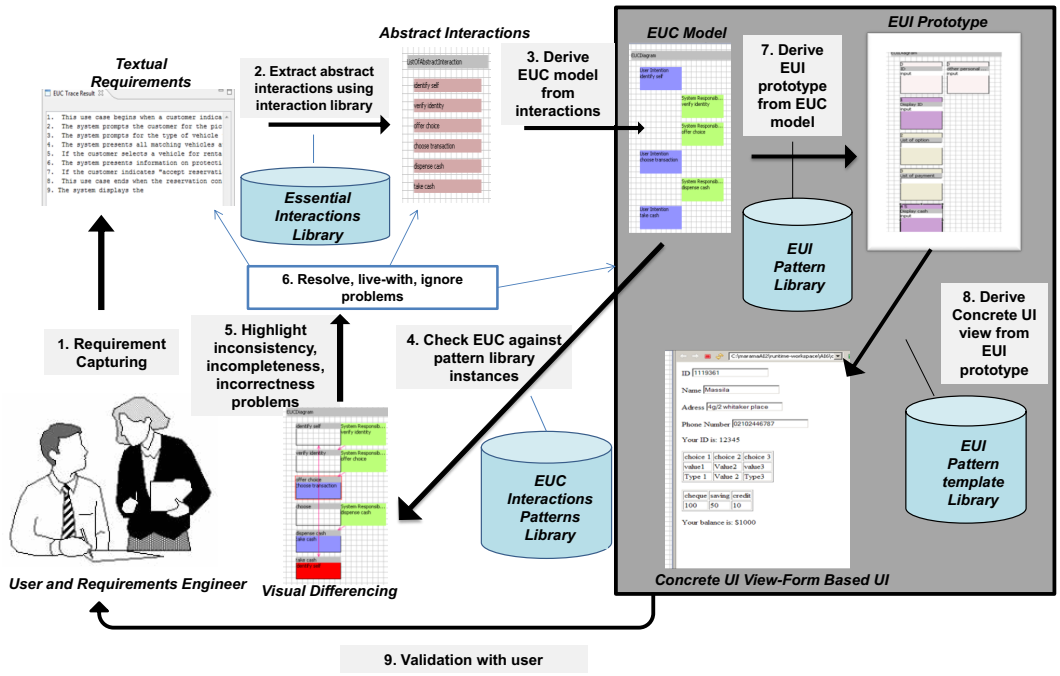


Fig 3. Usage of Marama AIC

Marama AIC improves traceability by implementing a lightweight approach together with a traceability technique and semi-formal specification in the form of EUC models in order to support consistency checking between the natural language requirement, an EUC model and an EUI prototype.

In Step 1, Requirements are first captured from natural language text.

In Step 2, Abstract interactions are then extracted using lightweight the natural language processing of phrase and regular expressions based on a essential interaction library, following the EUC approach described by Constantine and Lockwood and refined by Biddle et al.

In Step 3, an EUC model is then generated from the abstract interactions.

In Step 4, Extracted EUC models are checked against a repository of best practice EUC models derived from Biddle et al's work and our own. Sequences of EUC interactions can be compared to common sequences, or EUC interaction patterns, in our EUC interaction patterns library using this "differencing" concept.

In Step 5, Visual highlighting is used to warn the user of inconsistencies in any requirements element.

In Step 6, the requirements engineer can choose to resolve inconsistency, incompleteness and/or incorrectness problems detected, leave highlighted problem markers and later resolve them, or ignore problems until later.

In Step7, the tool also allows the requirements engineer to automatically and traceably transform EUC models to EUI prototypes using our novel EUI pattern library. This means traceability is provided throughout the process, allowing any of the EUI components to be traced forward/back from/to the EUC model, abstract interaction or textual natural language requirement.

In Step 8, MaramaAIC allows the EUI prototype to be translated to a more concrete form-based UI view, an HTML form, by using a novel EUI Pattern template library. An EUI prototype model can also be translated to a concrete form-based UI using a pre-defined template in a EUI pattern template library, with one template for each EUI pattern. Here, the EUI Pattern template consists of the descriptions of Concrete UI components to be instantiated for a particular EUI pattern. Simple interaction with the generated HTML form is also supported to illustrate how target system information input and output could work.

In Step 9, the EUI model and concrete UI generated from the tool can be reviewed by the requirements engineers with end-users to validate and confirm the consistency of the original textual requirements.

To achieve Steps 4 and 5 the extracted EUC model's abstract interactions are compared to an expected essential interaction and EUC pattern's set of abstract interactions and their sequencing. When any problems with requirements models are detected, the tool focuses on providing warning, feedback notification and visualisation of the quality issues existing in any component:

- Components that mismatch, do not exist in one model, have differing sequencing between components, or that overlap with non-corresponding names or other information, are classed as an "inconsistency".
- Detected redundancy of a component or a mismatch between a component and the expected element in an otherwise matching pattern is classed as "incorrectness".
- Missing components or sequences in a model compared to an otherwise matching pattern are classed as "incomplete". The set of requirements is assumed to be "complete" (Huzar, Kuzniarz, Reggio, & Sourrouille 2005) once all the requirements model elements satisfy a match or matches in the EUC interaction pattern library.

In Step 6, when any of the above problems are highlighted, requirements engineers then have the ability to choose to do one of the following:

I. Resolve a detected quality issue by modifying the components based on the results of the consistency engine recommendation.

II. Tolerate the inconsistency until later, with our tool tracking it.

III. Strictly ignore the inconsistency.

MaramaAIC avoids forcing requirements consistency immediately as consistency rules cannot always automatically maintain the consistency of the set of requirement components. For example, if the sequence of components of the abstract interaction or EUC is problematic, we cannot automatically enforce a change in the structure of the textual natural language as this requires manual intervention. In this situation, a warning and notational element highlighting make users aware that the inconsistency is present. Explicitly ignoring the inconsistency (suppressing warnings) is also allowed as it respects requirements engineers to make the final decision on the quality of their requirements. End-user stakeholders can view updated and/or annotated textual requirements at any time to understand the correctness and completeness of the requirements model. While the EUC model is arguably end-user-friendly, keeping it consistent with the textual natural language representation affords the latter human-centric views continued use through the requirements engineering process.

EUC and EUI Patterns Libraries

In order to simplify the above EUC and EUI extraction process, we adopted a domain-specific approach, instead of using conventional NLP-based approaches to capture requirements. This means we chose to develop a library of "proven" essential interactions expressed as textual phrases, phrase variants and limited regular expressions. We also developed a library of EUC patterns for higher level consistency checking and an EUI pattern library for the generation of the EUI prototype model (Kamalrudin, Grundy and Hosking 2011).

These libraries of essential interactions, EUC and EUI patterns were developed from a collection of such patterns previously identified by Constantine

and Lockwood (1999) and Biddle et al. (2000) together with patterns that were developed by us, which are all applicable across various domains.

Essential Interactions Patterns

We developed an essential interaction pattern library for storing essential interactions and abstract interactions. This essential interaction pattern library is based on a collection of phrases that illustrate the function or behaviour of a system. The collection of phrases is then categorised, based on its related or associated abstract interaction. We have collected and categorised phrases from a wide variety of textual natural language requirements documents available to us and stored them as essential interactions. Currently, we have collected over 360 phrases from various requirement domains including online booking, online banking, mobile systems related to making and receiving calls, online election systems, online business, online registration and e-commerce. The collection and categorisation of the phrases is an on-going process. Based on these phrases, we have come up with close to 80 patterns of abstract interaction. On average, there are 4.5 phrases or essential interactions associated with each abstract interaction. For example the abstract interaction "display error' is associated with four different essential interactions: "display time out", "show error", "display error message" and "show problem list". The essential interactions were not categorized based on one scenario. They have associations with up to five different concrete scenarios such as online business, e-commerce, online booking, online banking and online voting systems. One particular abstract interaction can be thus associated with multiple concrete scenarios. Table 1 shows some other examples of abstract interactions and their associated essential interactions for various domains of application.

Table 1 Example of Abstract Interactions and their Associated Essential Interaction and Their Related Domains

Abstract interaction	Essential interaction	Example of Domains
Verify user	verify customer credential	Online banking, online booking, online business, e-commerce, online reservation
	verify customer id	Online banking, online booking, online business, e-commerce, online reservation
	verify username	Online banking, online booking, online business, e-commerce, online voting system, online reservation
Ask help	help desk	Online banking, online booking, online business, e-commerce, online reservation

	request for help	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	ask for help	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	clicks help	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	complete help form	Online banking, online booking, online business, e-commerce, online voting system, online reservation
Offer choice	prompt for amount	Online booking, online banking, online business, e-commerce
	display account menu	Online banking
	display transaction menu	Online banking

In order to store the essential interactions in the essential interaction pattern library, selected phrases ("key textual structures") are extracted from the textual natural language requirement, based on their sentence structure. The 'key textual structure" uses Verb-Phrases (VP) and Noun-Phrases (NP) in the sentence structures to categorise the essential interactions. Any phrases that follow this structure will be acceptable as an essential interaction in the essential interaction pattern library. The tree structure of the key textual structure is illustrated in Figure 4. This shows that our library has three different sentence structures, based on the location of the Verb Phrase (VP) and Noun Phrase (NP). The Noun Phrase can contain structure elements such as Articles (ART) and Adjectives (ADJ) or only Nouns (Noun).

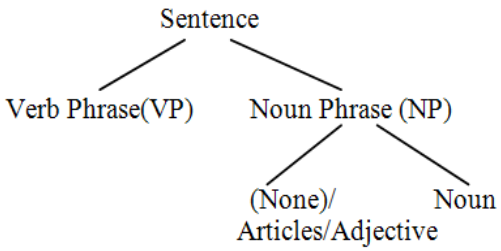


Fig 4. Tree Structure for Key Textual Phrase

The three different sentence structures are:

I. **Verb (V) + Noun (N) (only)** e.g. request (V) amount (N)

II. **Verb (V) + Articles (ART)+ Noun (N)** e.g. issue (V) a (ART) receipt (N)

III. **Verb (V) + Adjective (ADJ)+ Noun (N)** e.g. ask (V) which (ADJ) operation (N)

Below is an example of a part of textual natural language requirements by (Evan, 2009) that comprises the sentence structures of the key textual phrase to store the

essential interaction.

"The system **prompts the customer** for the pickup and returns locations of the reservation, as well as the pickup and return dates and times. The customer **indicates the desired locations and dates**. "

It is shown from the example that both underlined and bold requirements follow the key textual phrase of "Verb (VP)-Noun (NP)" but with different location of the Verb Phrase (VP) and Noun Phrase (NP). Both sentences of "prompts the customer" and "indicates the desired locations and dates" follow the second structure (II) of Verb+Article+Noun.

This key textual structure aims to provide flexibility in the library's ability to accommodate various types of sentences containing abstract interactions. With this, a broad range of phrase options can be extracted by the tracing engine, while still affording a lightweight implementation using string manipulation and some regular expression matching.

EUC Interaction Patterns

A set of best practice EUC interaction patterns or templates was developed based on a range of typical user/system interactions in a wide variety of domains (Biddle April 2000). The EUC interaction patterns library stores these best practice patterns of EUCs for each set of scenarios or use case stories. Table 2 illustrates some examples of EUC interaction patterns for scenarios such as "reserve item" and "purchase item", with their sequences of abstract interactions. We use these "best-practice" templates for higher level checking of consistency, correctness and completeness of a generated EUC model by comparing the EUCs to the templates.

Table 2 Examples of EUC Interaction Patterns

Scenarios/ Use Case stories	User intention Abstract Interaction	System responsibility Abstract Interaction
Reserve item	Choose	
		offer choice
		view detail
		request identification
	identify self	
		confirm booking
Purchase item	Choose	
		check status
	identify self	
	provides detail	
		verify identity
		request confirmation
		view detail

EUI Patterns and EUI Pattern Templates

We also developed a set of EUI patterns in an EUI Pattern library, using an adaptation of the brainstorming methodology proposed by Constantine and Lockwood (1999). This adaptation generalised their approach by providing a simpler and more generic EUI pattern for EUI prototypes. The generalised EUI pattern comprises four types of EUI pattern category: List, Display, Input and Action. These are similar to the concept of Containers, introduced by Constantine and Lockwood. The main aim of these EUI Patterns is to assist REs to rapidly model a user interface based on the requirements captured and modelled earlier in the EUC model. An abstract UI captured using such a pattern is used as a medium for early communication between the RE and the client as it is easy to understand and allows the client to narrow down UI detail before moving to the concrete UI. In more detail, the four EUI pattern categories are as follows.

- **List**: Show a list of items, options or values that are associated with a particular abstract interaction of the EUC model. Default values are provided from the UI pattern library but can be overridden during application.
- **Display**: Display output based on an associated abstract interaction of the EUC model. This could display a name, id, number, address, message or notification.
- **Input**: Allow a user to input data or details of a specific element associated with an abstract EUC interaction.

- **Action**: Show a control button, such as save, delete and submit, based on an associated EUC abstract interaction.

Each of these EUI patterns is associated with an abstract interaction from the EUCs. An EUI pattern can be associated with one or multiple abstract interactions. Table 3 shows some examples of mappings between abstract EUC interactions (right) and various EUI patterns (centre), and their categories (left). For example, the EUI pattern "Save" from the "Action" category is associated with three different abstract EUC interactions: "record call", "record detail" and "save identification". We can see that the abstract EUI patterns are very general and apply across a range of different domains. For example, the EUI pattern "Save" could support a range of different scenario domains such as making calls in a mobile application domain to online booking, registration and retail systems.

Table 3 Example of EUI pattern Category and its related EUI pattern and it's associated Abstract Interaction from the EUC model

EUI pattern category	EUI pattern	Abstract interaction
List	List of option	Choose
		offer choice
		Select option
	List of solution	offer alternative
		offer possible solution
	List of payment	choose transaction
		choose payment
		select amount
Display	Display payment	validate payment
		show payment
	Display Item detail	return item
		view detail
	Display status	check user
		Notify user
	Display ID	verify identity
		provide identification
	Display error message	display error
Input	ID	identify self
		request identification
	Other personal detail	identify self
		request identification
	Payment detail	make payment
	Item detail	provides detail
	Number	make call
		indicates number to dial
Action	Help	Ask help
		Present solution
	Save	record call
		Record detail
		save identification
	Print	Print
	Delete	delete item

The EUI Pattern template library is comprised of EUI Pattern templates which support translating the EUI prototype to concrete UIs in a form of HTML pages. An EUI pattern template is based on the EUI pattern used in the EUI prototype. The EUI pattern template is already pre-defined in the library. It contains templates defined in HTML format for each of the EUI pattern categories: List, Display Input and Action. The defined EUI Pattern template for the HTML form is as below;

i. List: Table
ii. Display: message/text/data/value
iii. Input: Text Input
iv. Action: Button

The EUI pattern template is also applicable and reusable for various domains of applications. Table 4 shows examples of EUI pattern templates with their associated EUI patterns and domains applicable to the pattern.

Table 4 Examples of EUI Pattern template with its associated EUI Pattern and associated Domains in the EUI Pattern template library

EUI pattern categories	EUI Pattern	EUI Pattern template	Domains
Action	Submit	Button	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	Add		
	Search		
List	List of item	Table	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	List of payment		
	List of option		
Display	Display availability	Numbers/text	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	Display amount	Value/text	
	Display ID	Numbers	
Input	Item detail	Text input	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	Payment detail		
	Problem form		

Tool Usage Example

In this section we illustrate the use of MaramaAIC using requirements which was developed by Evans and published on the IBM developer works website, as an example of a requirement to demonstrate the key features of our tool. This user scenario is a “hypothetical browser-based software system for an auto rental

company" (Evans, 2009) mainly for an individual account. It illustrates the situation that happens in a rental company when a customer comes to the rental counter to rent a vehicle (Evans, 2009). It is also an example from an online booking domain of application. The description of this user scenario is shown in Figure 5.

1. This use case begins when a customer indicates he wishes to make a reservation for a rental car.
2. The system prompts the customer for the pickup and returns locations of the reservation, as well as the pickup and return dates and times. The customer indicates the desired locations and dates.
3. The system prompts for the type of vehicle the customer desires. The customer indicates the vehicle type.
4. The system presents all matching vehicles available at the pickup location for the selected date and time. If the customer requests detailed information on a particular vehicle, the system presents this information to the customer.
5. If the customer selects a vehicle for rental, the system prompts for information identifying the customer (full name, telephone number, email address for confirmation, etc.). The customer provides the required information.
6. The system presents information on protection products (such as damage waiver, personal accident insurance) and asks the customer to accept or decline each product. The customer indicates his choices.
7. If the customer indicates "accept reservation," the system informs the customer that the reservation has been completed, and presents the customer a reservation confirmation.
8. This use case ends when the reservation confirmation has been presented to the customer.

Fig 5. Example of User Scenario: Reserve a Vehicle (Evans, 2009)

Example of Usage

Nancy, a requirement engineer, would like to validate the requirements that she has collected from the client, John, who is the car rental information manager. To do this, as shown in Figure 6, she types in the requirements in a form of user scenario to the textual editor or copies them in from an existing file (1) and has the tool trace the essential requirements (abstract interactions) (2). Here, she verifies the list of abstract interactions provided by the tool and then has the tool generate the EUC model (3). In order to check for the consistency and dependencies among the EUC component and the abstract interaction and the user scenario, she performs trace back by using the event handler from the EUC component or abstract interaction. For trace back (as shown in Figure 6), the selected EUC component (A)

and its associated abstract interaction (B) changes colour to red and the associated essential interactions (C) are highlighted with "***". The processes of tracing forward/backward and mapping are assisted by event handlers. These tracings show and maintain the consistency among the requirement components.

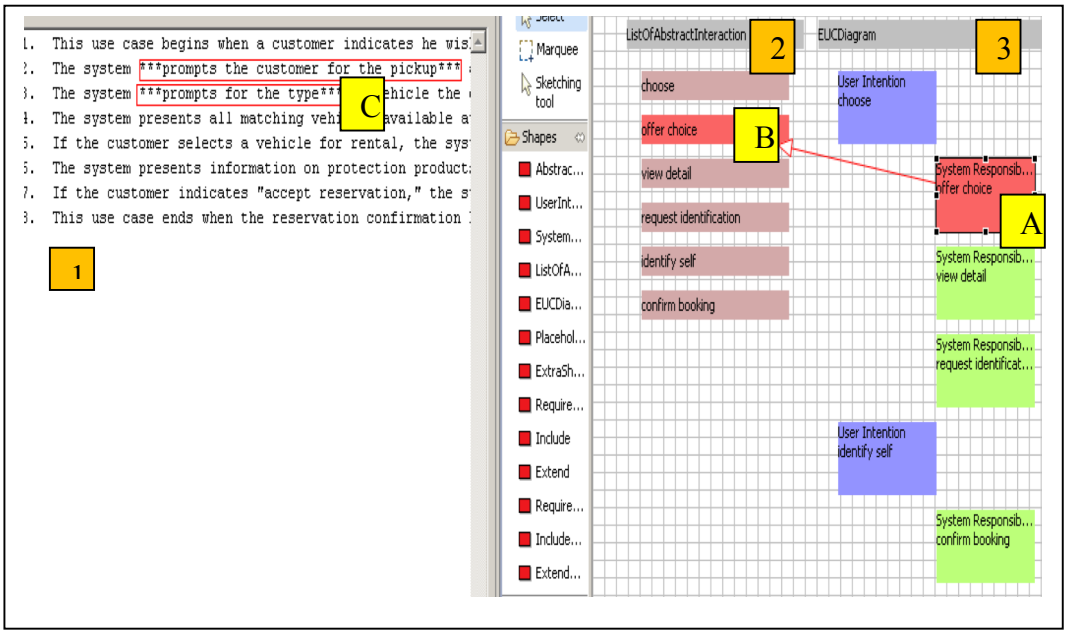


Fig 6. Capturing requirements - trace the abstract interaction, trace back and map to EUC model

By using MaramaAIC, Nancy can make any modification to any of the requirement components if she is not satisfied with the results provided by the tool. For example, if she thinks one of the abstract interactions is missing, she could add a new abstract interaction to the list. In particular, she might think that an abstract interaction "make payment" is missing from the list. Thus, she adds a new abstract interaction "make payment" to the list. This action triggers an inconsistency warning and the options either to update, delete or continue without updating the textual natural language requirements to appear to inform her that an inconsistency has occurred in the requirement components (as shown in Figure 7 (1)). She then chooses to continue without updating the user scenario as she probably thinks that the "make payment" abstract interaction is necessary and matches the user scenario. Although the option "continue" is chosen by her, she can still map the newly-added abstract interaction to the EUC model (2). This triggers a problem marker to inform her of the inconsistency error for later consideration to resolve the inconsistency (3).

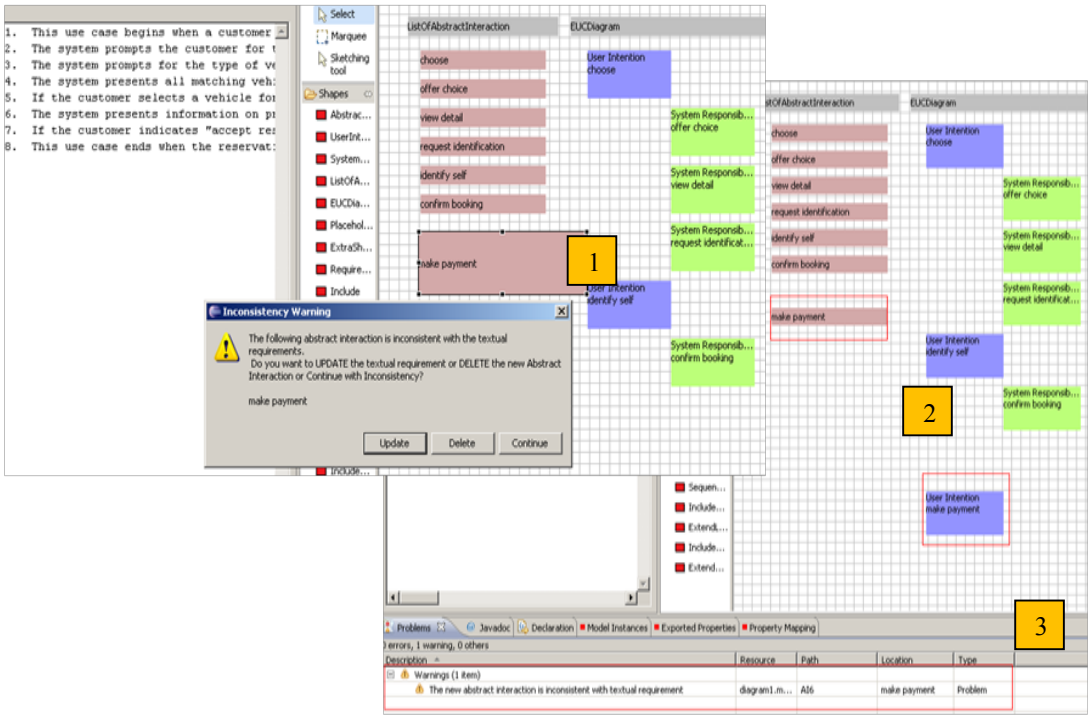


Fig 7. Add New Item to Abstract Interaction

Next, Nancy is also unhappy with the sequence ordering of one the abstract interaction components: “choose”. She thinks this abstract interaction should be above the “make payment” component as shown in Figure 8 (1) because the user should choose from the option before any payment should be requested. This triggers the associated EUC component “choose” to change colour to red and the essential interaction “indicates” to be highlighted with”***”. An Inconsistency warning also appears to inform her of the inconsistencies and provide options either to update or cancel the change. A problem marker also provides warning on inconsistencies that still exist. Then she decides to update the sequence ordering, and this automatically also changes the position of the EUC component” choose” (2). However, the ordering of the highlighted essential interactions is not altered as such changes could affect the structure of the user scenario. This action also triggers a problem marker to warn about the inconsistencies that have not been completely resolved.

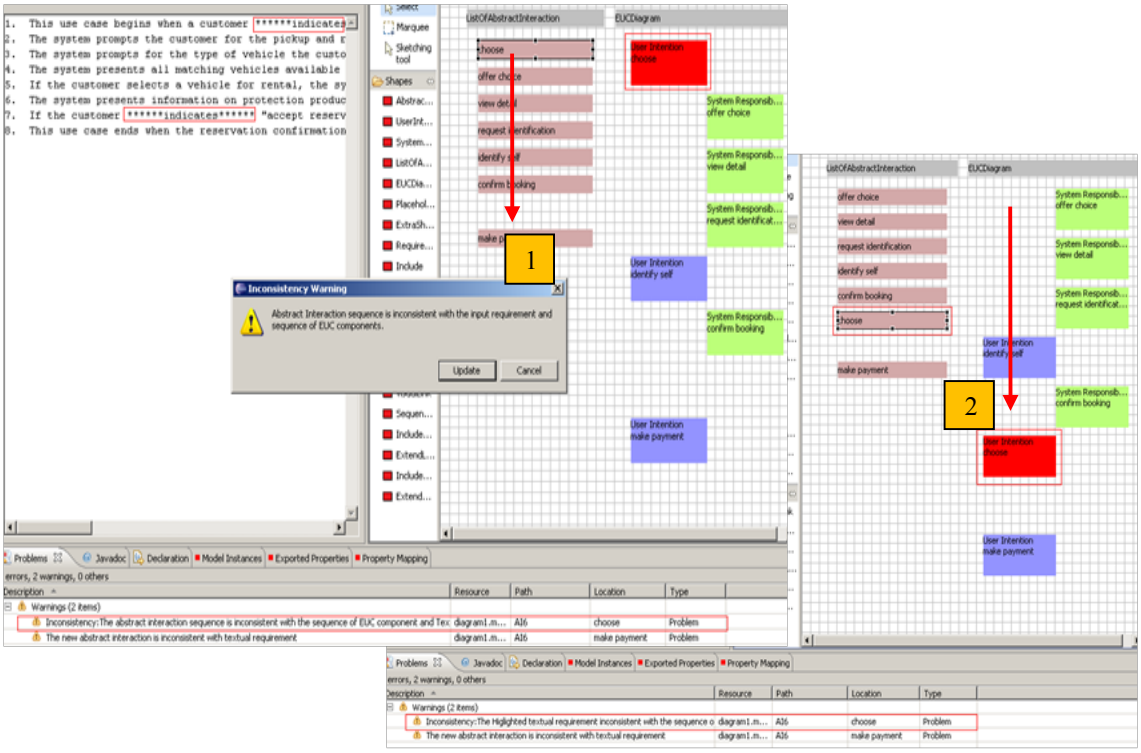


Fig 8. Change of Abstract Interaction Sequence Ordering

On reviewing the extracted EUC, Nancy feels that there is an extra component in the EUC model. She thinks that the EUC component "offer choice" is not necessary and needs to be deleted. She believes there is a redundancy between the "choose" and "offer choice" component as shown in Figure 9 (1). Thus, she selects the "offer choice" component to be deleted. This action triggers the associated abstract interaction to automatically change colour to red and the associated essential interactions "prompts the customer for the pickup" and "prompts for the type" to be highlighted with "***" as shown in Figure 9 (2). The inconsistency warning also appears to inform the inconsistencies and options to either delete or cancel the deletion. Although a notification of the inconsistencies is provided, she still thinks she needs to delete the "offer choice" component. This triggers the associated abstract interaction and essential interactions also to be deleted. This occurs as the tool tries to keep all the three requirement components in a consistent state.

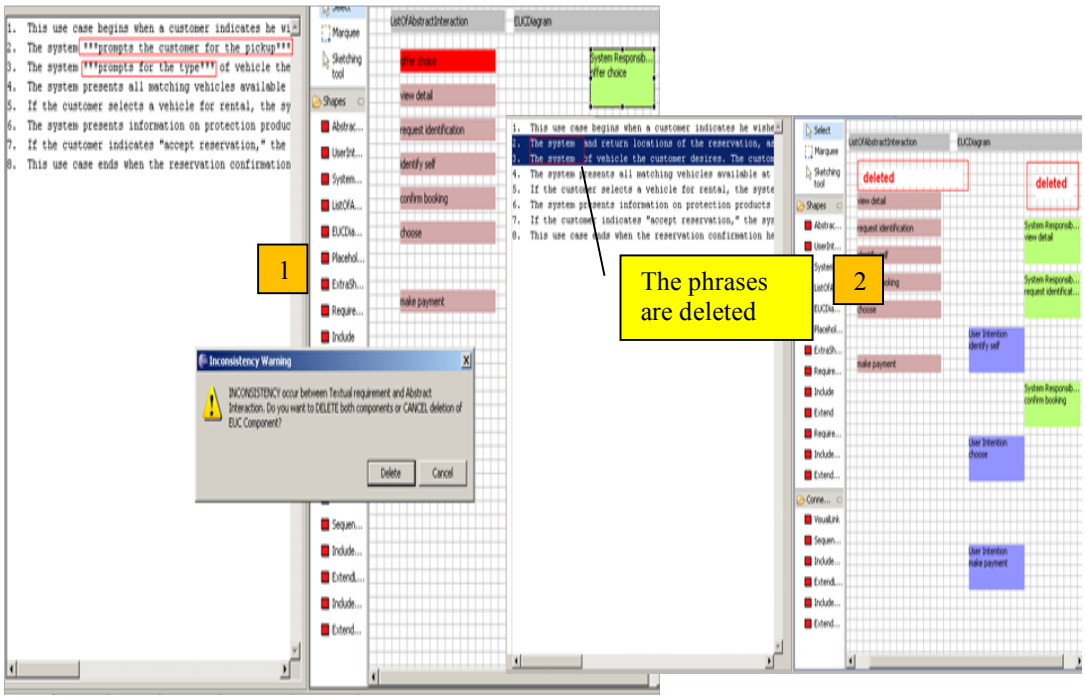


Fig 9. Delete the EUC component.

Being a novice requirement engineer, Nancy is keen to validate her extracted EUC model against a best-practice EUC template. Thus, she looks through the list of available templates and chooses the pattern "Reserve Item" as shown in Figure 10 (1) that appears to be similar to this scenario. She matches the pattern to her EUC model and sees that she has missed some interactions as a few sequence orderings and components are incorrect. In addition, an extra component also exists in the interaction. As shown in Figure 10 (2), the incorrect sequence ordering is shown by the red visual links (A), the existence of the extra component "make payment" (B) is outlined with red and the correct component "offer choice" (C) is shown by a grey element on top of the green shape "view detail" which also displays the incorrect component and position held by the "view detail" component. As there is an unmatched interaction between the generated EUC and the best-practice template, Nancy is notified with an inconsistency warning and given options to either keep or change the generated EUC following the best-practice template. She agrees with the warning and the errors shown. She then selects to change this EUC model to the EUC interaction templates.

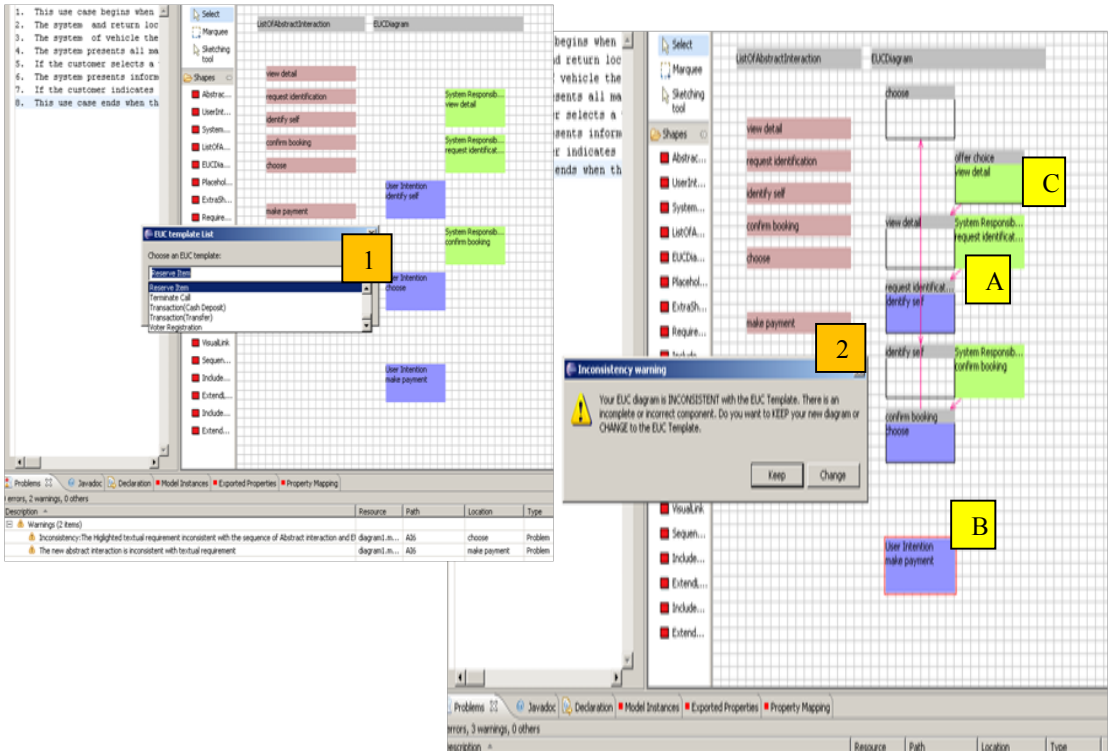


Fig 10. Visual differencing to check for incorrectness and incompleteness

When Nancy is satisfied with the requirements components, she sits with John to validate the requirements and to confirm the consistency of her captured requirements with the earlier requirements provided by John. In order to allow John to better understand the requirement components, she then has the tool map the EUC model to abstract prototype: EUI prototype as (1) and also has the tool translate EUI prototype to a concrete UI view in a HTML form (2) as shown in Figure 11.

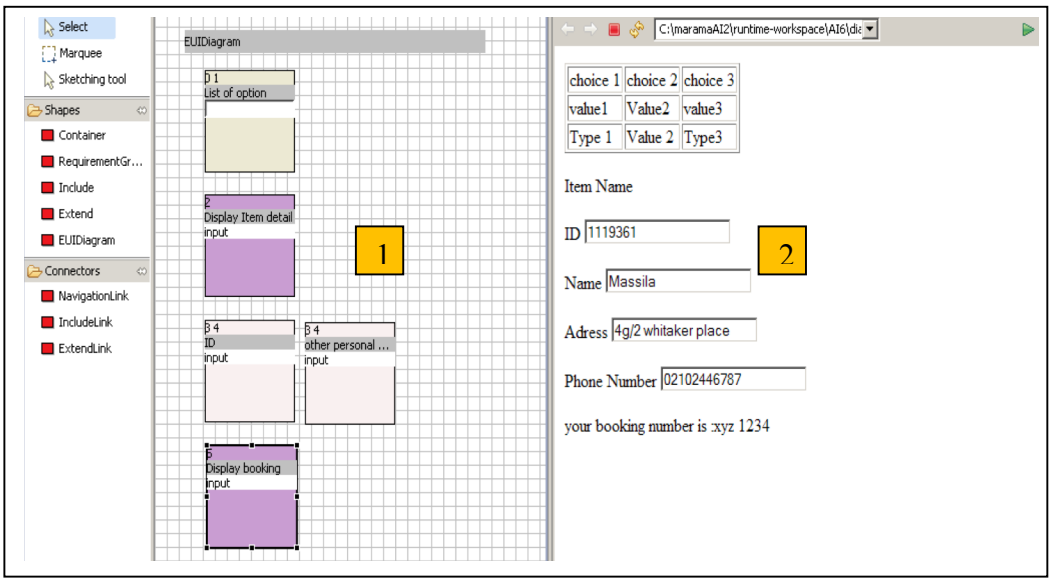


Fig 11. The generated EUI prototype (1) and translated HTML form (2)

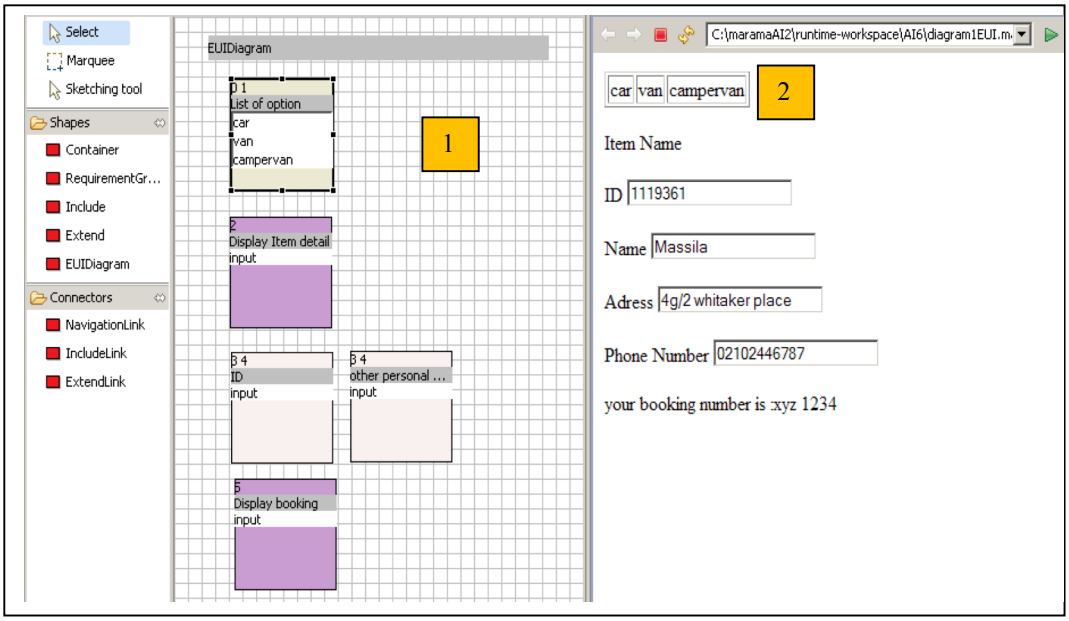


Fig 12. Modifications in Prototypes

From the walkthrough, John thinks that the EUI component of "List of options" is a bit vague and would be better understood by adding detail of the types of options such as "car, van and campervan" as shown in Figure 12 (1). Nancy modifies that on the spot and then shows the result in a HTML form as in Figure 12 (2). Next, she wants to validate and confirm the consistency of her point of view against John's point of view. She selects one of the EUI components "List of options" (A) and has the tool trace back to the other requirement components: EUC

model, abstract interactions and textual natural language requirements as shown in Figure 13. This triggers the associated EUC component and abstract interactions "choose and offer choice" (B) to change colour to red and the essential interactions "indicates, prompts the customer for the pickup and prompts for the type" (C) of the user scenario to be highlighted. Here, Nancy is able to confirm the consistency of all requirement components with John for the earlier collected requirements.

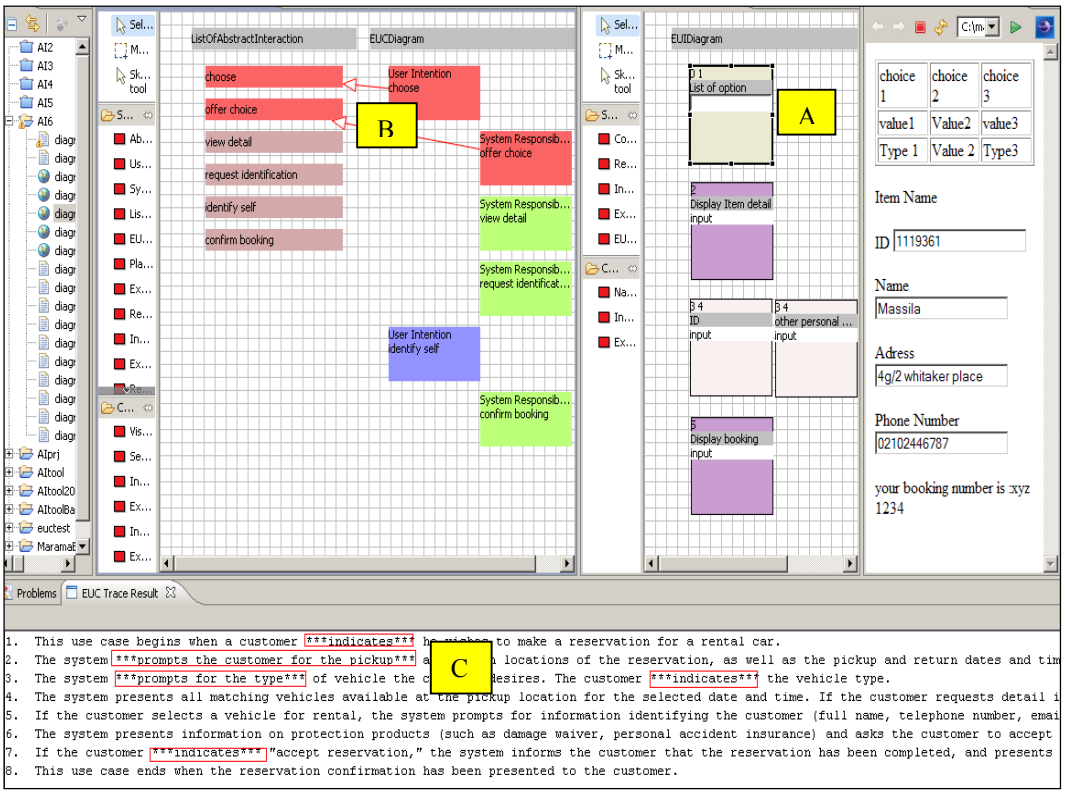


Fig 13. Trace back which performs from the EUI prototype

In summary, Nancy has used the MaramaAIC tool to capture automatically the abstract interactions and to extract the EUCs from the user scenario provided by John. She also used the tool to manage the consistency and to validate the incorrectness and incompleteness of the requirements by using the essential interaction pattern library and "best- practice" template from the EUC interaction pattern library, together with the inconsistency warning, problem marker and highlights. She then sat with John to verify and confirmed further the consistency of the requirements by having the tool generate the prototypes: EUI prototype and HTML form.

Architecture and Implementation

MaramaAIC consists of textual natural language requirement, abstract interaction, Marama Essential (EUC diagram) and MaramaEUI (EUI prototype model) editors. The architecture of Marama AlC is shown in Figure 14. MaramaAIC was realised using the Marama meta-toolset (Grundy et al. 2008) , which is built using the Java–Eclipse platform [Steps 1-2 in Figure 12]. MaramaAIC editors are specified using Marama shape, meta-model and view tools. Each editor is then implemented by interpreting the specification using a set of Marama plug-ins [Step 2]

The meta-model and Domain Specific Visual Language (DSVL) specifications were also supplemented with event handlers to provide low-level model constraints, consistency management support, mapping and interfaces to other elements of the architecture as well as to generate the prototype model (3-7). These were implemented in Java and include generation of dialogues and problem markers to help the user to track, tolerate and resolve the inconsistencies. The event handlers are the vital agent in maintaining consistency among the four forms of requirements components: textual natural language requirements, abstract interaction, EUC diagram and EUI prototype model. An Eclipse text editor is used to capture natural language requirements and "event handlers" [Step 3] called Trace were implemented to realise extraction of abstract interactions and EUC models from the natural language text. This EUC extractor generates an editable Marama EUC diagram. An MS Access database of mappings of essential interactions to abstract interactions is used in this extraction process. Source natural language phrase to EUC element mappings are recorded with the EUC elements during the extraction process. This allows tracing between these elements when the MaramaAIC user clicks on an item in each view. The "trace back" event handler [Step 3 and 7] uses these mappings to visually highlight the linked natural language phrases, EUC elements and EUI prototype respectively.

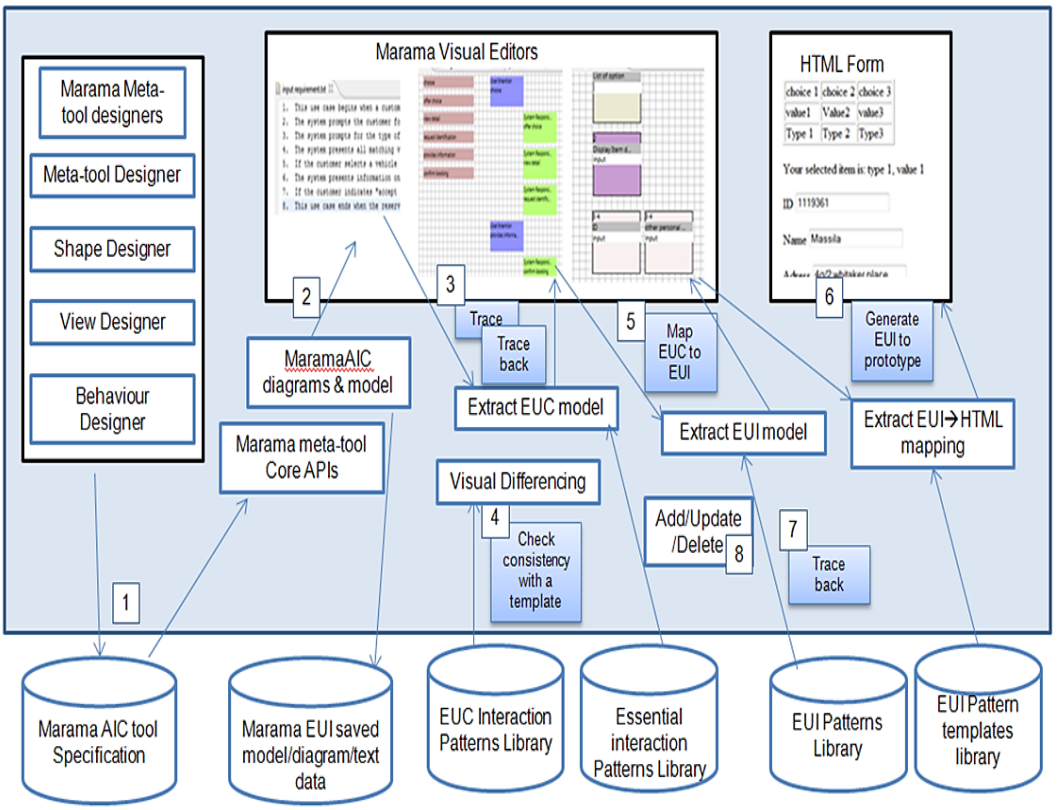


Fig 14. Architecture of MaramaAIC.

A "visual differ" [Step 4] is used to highlight the differences between "best-practice" pattern template and EUC. This often highlights incomplete and/or incorrect sequences, elements, missing elements, or mistyped elements in the extracted EUC, helping the MaramaAIC user to identify problematic requirements.

Another event handler generates an EUI model from the EUC model [Step 5]. This uses a EUI pattern library to map EUC elements to best-fit EUI elements. This EUI model can then be used to generate an HTML form representing a rapid prototype of a form-based interface to the requirements [Step 6]. Updates to any of the models (natural language, abstract interactions, EUC elements or EUI elements) are detected as they are made [Step 8]. These changes are propagated to related elements in the other models. Some changes can be automatically applied. Others are ambiguous so the tool informs the user of the change(s) so the user can make appropriate manual updates. To illustrate further how the event handlers work in our tool, sequence diagrams are used to demonstrate the interaction. Figure 15 and Figure 16 show an example of interaction of TraceBack and IndexChecker event handlers in operation.

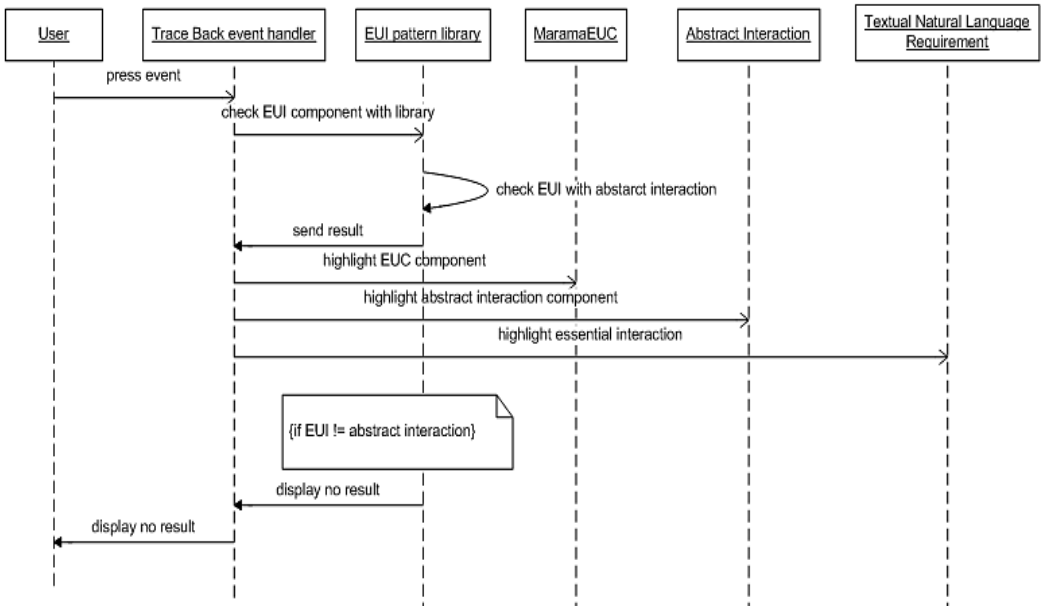


Fig 15. Example of TraceBack interaction from EUI prototype to EUC Model

Figure 15 shows how the user traces back from the EUI prototype component to its source using the TraceBack function. The selected EUI prototype component is analysed by the tracing engine and then matched with the abstract interaction in the EUI Pattern library. If we try to trace back the EUI component, the tool will show where the associated abstract interaction, EUC model and essential interaction for that particular EUI prototype come from. If a newly added component of the EUI prototype does not match an abstract interaction in the EUI Pattern library, no result is provided.

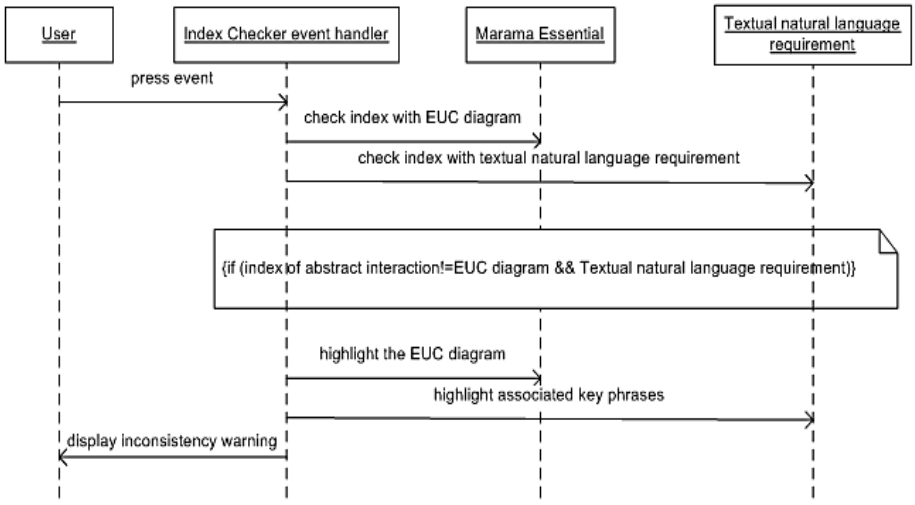


Fig16. Example of Index Checker interaction of Abstract Interaction

Figure 16 shows the function IndexChecker which acts as a checker for the consistency of the sequences in both abstract interaction and EUC Diagrams in Marama AIC. 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 natural language requirements. If there is any change of the sequence or location for either, the event handler highlights the associated components either the EUC component or the essential interations and provides a warning about the inconsistency that has occurred.

Evaluation

Recall that our aim of this work was to determine whether automated tool support for EUC-based requirements capture and validation would improve on manual methods, captured by our research question of "***can automated support for Essential Use Case and Essential User Interface modelling enhance the consistency management and validation of requirements over manual methods?".***

We conducted three studies in order to evaluate our tool's efficacy, performance and usability. The first study was on the efficacy and performance of our tool to extract abstract interaction for EUCs. Results were then compared with the preliminary study on the manual extraction process.

The second study was of our tool's usability and user perceived strengths and weaknesses. Here, participants explored the tool facilities for capturing and checking the consistency of requirements as well as generating the prototype model. The second study was conducted individually to allow us to observe participants and receive feedback one-on-one from them. Participants were given an explanation and demonstration of how to use the prototype tool and the tasks they needed to perform. A task list and a questionnaire sheet were given to participants before they started using the prototype tool.

The third, qualitative, study evaluated the effectiveness of our end-to-end rapid prototyping approach in improving the dialogue between REs and their clients and in improving requirements' quality. Here, we interviewed and observed three pairs of industry practitioners, one pair member is an industry-based software practitioner experienced in handling software requirements and the other an industry based practitioner experienced in the role of being a client or stakeholder

in a software project. This study aimed to understand whether the tool was effective in supporting round trip requirements engineering and validation between REs and their clients.

Efficacy evaluation

We first compared the accuracy of MaramaAIC against the previous results of preliminary study on the manual extraction of essential requirements by 11 novice requirements engineers, as shown in Table 5. MaramaAIC succeeded in identifying almost all the abstract interactions, failing to detect one abstract interaction, providing an accuracy of almost double the participants' average and better than all but one of the participants' accuracy. The correctness ratio for manual extraction is only 47% while MaramaAIC's is 83%. The single error from the tool is because of its failure to detect one of the abstract interactions (Take Cash).

Table 5 Comparison of Manual Extraction and Automated Support of MaramaAIC

Answers	No. Correct answers	
	Manual extraction	Automated Tracing
Identify user	5	1
Verify Identity	4	1
Offer cash	4	1
Choose	6	1
Dispense cash	9	1
Take cash	3	0
Correctness ratio	47%	83%

In order to determine the scalability and efficacy of our tool, we further evaluated its accuracy by applying it to extract EUC models for 15 use case scenarios derived from different researchers, developers and ourselves across a variety of different domains: Online CD catalogue, Cellular phone (Constantine 1998), Voter registration (Stephane 2005) Cash withdrawal (Bjork 2005) Online book (Glinz 2000), Checkout book (library) (Denger, Berry & Kamsties 2003), Seminar Enrolment (Nuseibeh, Easterbrook & Russo 2000), Transfer transaction (Bjork 2005), Deposit transaction ((Bjork 2005), Assign report problem (Horton 2009), Create problem report (Horton 2009) , Report problem (Horton 2009),

Booking room (Kim 2006) and Place order (Scenario examples, 2009). The tool correctness was evaluated by comparing the answers with oracle EUC models provided by Constantine and Lockwood (1999), Biddle et al. (2002) and also with models we developed following Constantine and Lockwood's methodology. Correctness ratios for the abstract interactions identified, calculated as they were for the manual extraction study, are shown in Table 6.

Table 6 Efficacy Evaluation on the Extraction Process using MaramaAIC

No.	Requirement	Numbers of Abstract Interaction	Manual results: List of abstract interaction	Automated results: List of abstract interaction	Numbers traced	Ratio
1	Online cd catalog	5	1.view list	✓	5	5:5
			2.search item	✓		
			3.view details	✓		
			4.make order	✓		
			5.calculate cost	✓		
2.	Cellular phone	3	1.make call	✓	2	2:3
			2.receive call	x		
			3.answer call	✓		
3.	Cash withdrawal	6	1.choose account type	✓	4	4:6
			2.select amount	✓		
			3.verify amount	x		
			4.view problem	✓		
			5.verify transaction	✓		
			6.notify result	x		
4.	Online book	7	1.select item	✓	6	6:7
			2.make payment	✓		
			3.ask help	✓		
			4.notify confirmation	x		
			5.verify user	✓		
			6.print invoice	✓		
			7.sent item	✓		
5.	Voter registration	6	1.select option	✓	6	6:6
			2.request identification	✓		
			3.identify self	✓		
			4.check status	✓		
			5.provide identification	✓		
			6.display error	✓		
6.	Borrow book	7	1.verify user	✓	3	3:7
			2.display option	x		
			3.select option	x		
			4.check item	✓		
			5.identify item	x		
			6.print slip	✓		
			7.display message	x		
7.	Checkout book(library)	6	1.identify user	x	5	5:6
			2.verify user	✓		
			3.validate item	✓		
			4.print receipt	✓		
			5.receive receipt	✓		
			6.return item	✓		
8.	Enrollment seminar	9	1.identify self	✓	8	8:9
			2.verify user	x		
			3.display option	✓		
			4.make selection	✓		
			5.check the schedule	✓		
			6.calculate cost	✓		
			7.enroll	✓		
			8.ask payment	✓		
			9.print bill	✓		

9.	Transfer transaction	6	1.select option	✓	6	6:6
			2.chooses account type	✓		
			3.select amount	✓		
			4.provide identification	✓		
			5.verify user	✓		
			6.print receipt	✓		
10.	Deposit transaction	6	1.select option	✓	6	6:6
			2.chooses account type	✓		
			3.select amount	✓		
			4.provide identification	✓		
			5.verify user	✓		
			6.print receipt	✓		
11.	Assign report problem	4	1.select option	x	2	2:4
			2.display result	✓		
			3.select member	x		
			4.confirm status	✓		
12.	Create problem report	7	1.select option	x	4	4:7
			2.request report	x		
			3.create report	x		
			4.save identification	✓		
			5.confirm status	✓		
			6.insert description	✓		
			7. save report	✓		
13.	Report problem	6	1.identify self	x	5	5:6
			2.display help	✓		
			3.select help option	✓		
			4.request description	✓		
			5.describe problem	✓		
			6.offer possible solution	✓		
14.	booking room	4	1.select option	x	3	3:4
			2.select item	✓		
			3.identify self	✓		
			4.print slip	✓		
15.	Place order	6	1.identify self	x	4	4:6
			2.select product	✓		
			3.provide detail	✓		
			4.make payment	✓		
			5.verify information	✓		
			6.confirm order	x		

This shows some variability across the range of scenarios, averaging approximately 80% correctness for extracting abstract interactions. The automated tracing tool does not (and cannot) produce 100% correct answers due to the inherent incorrectness and incompleteness of textual requirements. This is due to various linguistic issues, such as phrases or sentences using a passive pattern, existence of parentheses and grammar issues such as incorrect use of plural or singular, adjectives or adverbs (Tjong, Hallam & Hartley 2006). These problems, however, also lead REs to misunderstand requirements and can be one of the reasons why different requirements engineers or users provide inconsistent results. An average 80% extraction accuracy is lower than desirable, however two points need to be made. Firstly, the accuracy is much better than for manual extraction. Secondly, many of the inaccuracies are picked up when the extracted EUC models are

matched against best practice EUC patterns in downstream use of the toolset. This, in turn, can help, via use of the MaramaAIC traceability tooling support, to identify grammatical problems with the textual requirements that cause inaccurate extraction of EUC elements.

Usability study

In our preliminary study, we demonstrated that end users find manual derivation of EUC and EUI prototypes to be difficult, time consuming and error prone. We wanted to demonstrate the effectiveness of our new automated tool support using EUC modelling and EUI prototyping together to support end-to-end rapid prototyping consistency management and validation of requirements. To this end we conducted a user study to evaluate perceptions of the tool and its application.

Participants in this study were 20 software engineering post-graduate students. Their experience as requirements engineers can be categorised as novice to intermediate. Each participant was given a brief tutorial on how to use the tool and some examples of how the tool captures requirements using EUC modelling and EUI prototyping. They first captured the requirements using EUC models and then derived an EUI prototype from the EUC model and natural language requirements. They then mapped the EUI prototype to a concrete HTML-based UI view. Further exercises modifying the EUI prototype followed: adding and deleting EUI components and exploring the result of the modifications in the concrete UI view. We observed the participants' performance while using the tool to accomplish the provided tasks. Participants were asked to think aloud and provide suggestions to enhance the tool. Once all tasks were completed for each part, they were required to answer a questionnaire. Participants completed the questionnaire at their own pace without supervision. The response data were then collected for analysis. Each participant took less than one hour to perform the evaluation. The questionnaire comprised two parts, examining 1) usability and 2) a Cognitive Dimensions (CD) (Blackwell 2001) based assessments. Each question was recorded using a five part Likert scale: 1=strongly disagree to 5=strongly agree.

For Usability criteria, we used the set of criteria suggested by Lund (1998) in the USE questionnaires. The author suggested four criteria that are correlated to one another - Usefulness, Ease of Use, Ease of Learning and Satisfaction (Lund 1998).

We used these criteria in developing our questionnaires. We define the criteria as follows.

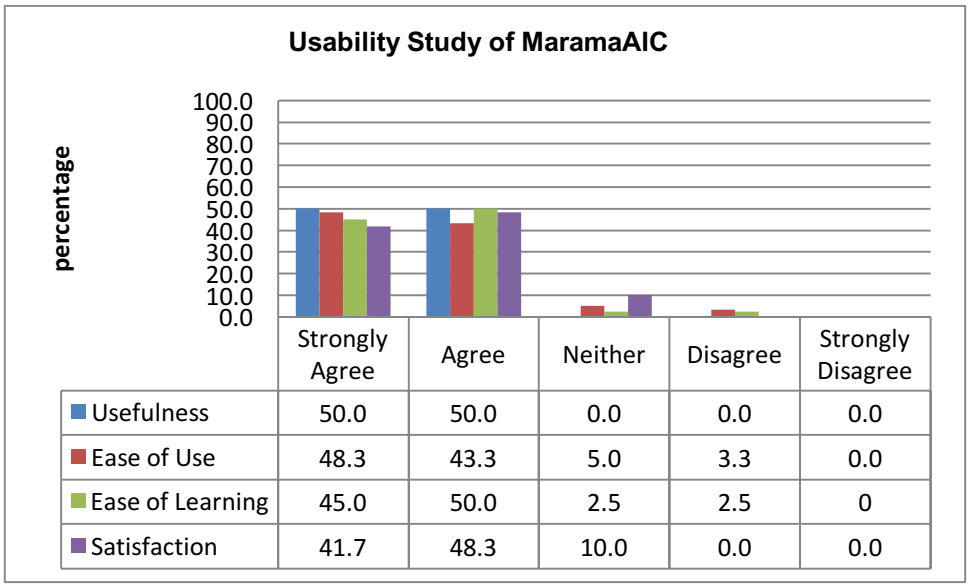
- Usefulness: how useful the tool is to help users be effective in accomplishing the given task
- Ease of Use: how easily users can work with the tool's user interface and functionality
- Ease of Learning: how easily the user can understand and learn to use the tool
- Satisfaction: is the user satisfied with the tool's capability in performing the required tasks.

The questionnaire comprised several questions for each criterion, which were averaged and converted to a percentage.

We used the Cognitive Dimensions (CD) framework operationalised by Blackwell (2001) in our questionnaires to allow us to explore in detail the reason for each of the user's perceptions for our MaramaAIC tool. CD (Blackwell et al. 2001) is applied here, as it is a common approach for evaluating visual language environments. It helps non-HCI specialist and ordinary users to evaluate usability (Blackwell 1998). In addition, it is lightweight and allows reasoning about usability tradeoffs (Blackwell 1998). In our questionnaire each CD dimension was evaluated by one question. The questions used are adapted from (Kutar 2000). In total, there were ten questions as shown in Table 7.

Table 7 CD Notations Used and Questions Evaluating Them

Cognitive Dimension	Question
Visibility	It is easy to see various parts of the tool
Viscosity	It is easy to make changes
Diffuseness	The notation is succinct and not long-winded
Hard mental effort	Some things do require hard mental effort
Error-proneness	It is easy to make errors or mistakes
Closeness of mapping	The notation is closely related to the result
Consistency	It is easy to tell what each part is for when reading the notation
Hidden dependencies	The dependencies are visible
Progressive evaluation	It is easy to stop and check my work so far
Premature commitment	I can work in any order I like when working with the notation

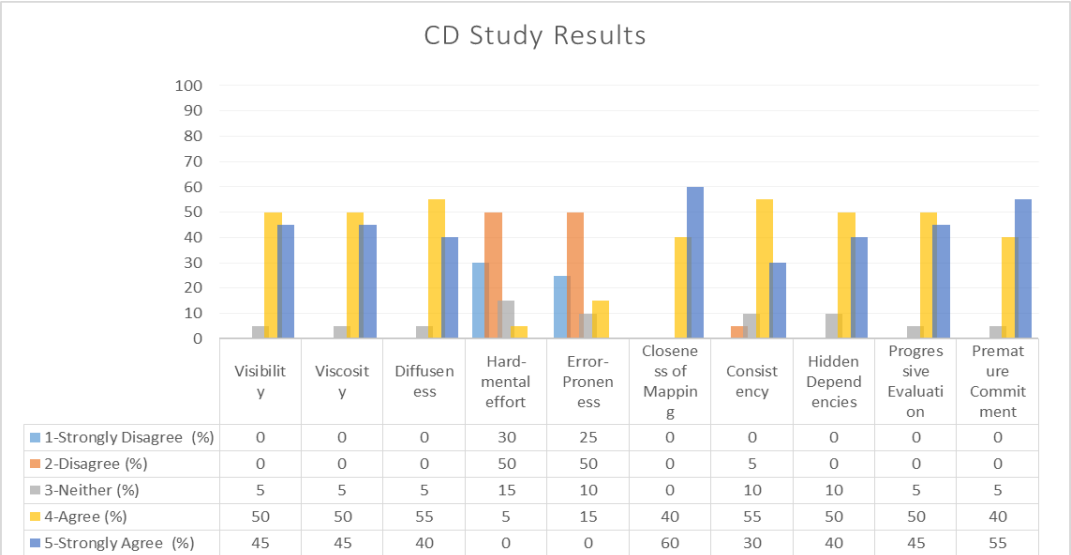


	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree
Usefulness	50.0	50.0	0.0	0.0	0.0
Ease of Use	48.3	43.3	5.0	3.3	0.0
Ease of Learning	45.0	50.0	2.5	2.5	0
Satisfaction	41.7	48.3	10.0	0.0	0.0

Fig 5. Usability results on MaramaAIC

Figure 17 shows the results of the usability survey conducted for MaramaAIC. For each characteristic, the results of each corresponding question block were averaged to produce the results shown. The results are overall positive with strong agreement from the users over the usefulness of the tool (100% strongly agree or agree on its usefulness), the ease of use (over 90%), ease of learning (95%) and satisfaction (90%). The small number of cases of disagreement over ease of use and ease of learning related to a preference by those participants to have a more descriptive label for each colour and shape used in MaramaAIC. However, with the small number of experimental subjects the results should be viewed as encouraging but not definitively answering our research question.

The CD study allows us to explore in more detail the reasons for these user perceptions. We used the dimensions and questions in Table 7 for this study. The results are based on percentages, reflecting the number of participants' answers for each scale. Figure 18 shows the evaluation results for each of these questions. We believe these results demonstrate interesting usability dependencies between the dimensions that we feel have contributed to the strong usability acceptance of our MaramaAIC.



	Visibility	Viscosity	Diffuseness	Hard-mental effort	Error-Proneness	Closeness of Mapping	Consistency	Hidden Dependencies	Progressive Evaluation	Premature Commitment
1-Strongly Disagree (%)	0	0	0	30	25	0	0	0	0	0
2-Disagree (%)	0	0	0	50	50	0	5	0	0	0
3-Neither (%)	5	5	5	15	10	0	10	10	5	5
4-Agree (%)	50	50	55	5	15	40	55	50	50	40
5-Strongly Agree (%)	45	45	40	0	0	60	30	40	45	55

Figure 18 CD Study Results

A summary of the results for each dimension are as follows:

Visibility: Visibility was rated highly, due to explicit use of juxtaposition and the visible trace links. Our speculation is that this is because this "completes the picture" for users.

Viscosity: Participants found it is easy to make changes to the diagrams either in the EUC model or EUI prototype model.

Diffuseness: The notations used by MaramaAIC are succinct and understandable by end users.

Hard-mental effort: MaramaAIC does not need a lot of mental effort to solve the tasks. The tool is able to automatically detect inconsistencies in the requirements and automatically generate the various models.

Error-Proneness: Users disagree that the tool leads the user to errors. This is because all the errors are detected automatically and they could automatically generate the prototype. The EUC and EUI prototype generated is based on the pre-defined pattern library. Thus, this assures the accuracy of the EUC and the UI

Closeness of Mapping: The notations used by the tool are relatively intuitive and understandable. However, the Marama layout mechanism sometimes confused users as to which notation was being used when doing visual diffing.

Consistency: Some of the users were confused when differentiating the notation used to represent the differences between the generated EUC model and the EUC template models.

Hidden dependencies: This rated highly as dependencies among the three requirements components and prototype are made visible using highlighting.

Progressive Evaluation: This rated highly. MaramaAIC allows end users to easily stop and check their work at any time and to changes to be made to any of the requirement components. Thus, end users do not have to worry about the errors as the tool provides an automated support if any errors such as inconsistencies, incompleteness and incorrectness exist.

Premature Commitment: This dimension, which also rated highly (i.e. users regarded the system as having low premature commitment), reflects the sequence of using the tool in order to achieve the results. The tool allows a user to perform the task from any direction. End users can capture requirements or make changes in any of the components either from the MaramaEUI editor or MaramaEssential editor with consistency maintained.

To summarise, the usability dependencies between the dimensions show that high closeness of mapping and visibility as well as low viscosity assists with issues of hard mental operations and hidden dependencies but somewhat surprisingly did not reduce participants' impressions of error proneness. The high progressive evaluation and low premature commitment contribute to low viscosity.

Use of MaramaAIC by Requirements Engineering Professionals

In our third evaluation, we conducted a qualitative study using pairs of participants, one an industry-based software practitioner experienced in handling software requirements and the other an industry based practitioner experienced in the role of being a client or stakeholder in a software project. Three pairs of participants were recruited. Table 8 shows the background of the participants involved. Pairs of participants were given an explanation and demonstration of our MaramaAIC tool and some requirements extraction, tracing, consistency checking and UI prototyping tasks to be performed.

Table 6 The Participants' Background

Evaluation	Participants (RE==Requirements Engineer, C==Client)	Position	Level of Experience in Requirements	Years of experience in Requirements	Client Background
1	RE 1 & C1	Software Engineer	Intermediate	5 years	Government client with IT Background
2	RE 2 & C2	Staff engineer	Advanced	5 years	Private client with IT Background
3	RE 3 & C3	Senior System Analyst	Advanced	4 years +	Private client with IT Background

Each participant needed to capture textual requirements from the client, map these to an EUC model, and then map them to a UI rapid prototype model. They then showed the results of this to the client participant. Any changes or modifications requested by the client were carried out by the RE using MaramaAIC. We observed the participants carrying out these tasks and video-recorded them to enable us to more closely analyse how the tasks were performed. The participants were also asked to think aloud and express their opinions about the tool. At the end of the session they were asked to answer questions in a semi-structured interview covering the topics of whether the approach helped to improve the dialogue between the RE and client and whether it helped to improve requirements quality.

From our observations and interviews, we found that MaramaAIC assisted both REs and clients to discuss, to confirm and to validate the target system requirements. In evaluation 1, RE1 stated that the tool encouraged her to ask the client to confirm and validate the consistency and correctness of the requirements that she had captured in EUC model. An extract from the dialogue is as follows:

RE1: “So, here is the picture of your requirements. What do you think?”

C1: “All looks good but this component (“list of option”) is not necessary.”

RE1: “Ok. Let's delete the “list of option” and let us see the prototype.”

RE1 deleted the component as requested and then showed C1 the textual requirements and EUC, EUI prototype model and HTML form generated before. C1 is then requested to validate and confirm the modified requirements against the original requirements and responded:

C1: “yes. I think it is fine now.”

A similar dialogue occurred in evaluation 2:

RE2: “This is the outcome of your requirements. Can you please have a look on the prototype to confirm that I’m on the right direction.”

C2: “I think something is not right here. I think I need to add a component (“list”) to the prototype. Can you please show me the original requirements that I gave before?”

RE2 showed C2 the original requirement in the textual editor written in NL and then made changes by adding the component “list” as requested and then asked C2 to validate the modified requirements.

RE2: “Here are the original requirements and this is the result of the new one. I think this component (“list”) is not right to be here. Do you still want it to be added?”

C2: “I think you are right. Delete the list and keep the requirements as it is.”

In the case of evaluation 3, RE 3 stated that the tool helped her to visualise the interaction and the outcomes of her captured requirements via the EUC model and prototype model with the client (C3). A dialogue similar to the previous two eventuated:

RE3: “Cool! The tool shows me the interactions between user and system and the prototype. So, sir, here is the picture of your requirements. What do you think? ”

C3: “Cool. But I think I need to add a button (“delete”) here”.

C3 asked the RE to add a component (delete) at the end of the page. RE3 made the changes as requested.

RE3: “Ok. Let's see if it fits with your original requirements (while tracing it back to the textual requirements). It seems fits well here. I think I agree with you”

C3:”Thank you. Everything is perfect now.”

In all three cases the tool helped both clients and REs to check the consistency, correctness and completeness of the requirements against the client’s

original intentions allowing them in real time to explore, discuss and agree or disagree with changes made to the requirements. MaramaAIC helped to both ease and speed up the process of requirements validation through its fast feedback on the impact of changes or modifications.

Overall, the evaluation and interviews with the participants provided positive results. All the REs stated that that the tool helped them to communicate and discuss uncertainty and problems with the clients as well as to confirm and show the results of the requirements to the clients. They were also happy with the explanation and arguments from the clients as they could visualise the results using the prototype. They commented that they did not need to wait for a long cycle of meetings with clients to confirm requirements. Client participants all agreed the tool helped them to clearly identify any errors and misunderstandings and communicate them to the RE. They liked the fact that changes were able to be made immediately and their effects visualised at the same time. This gave them confidence that their requirements were correct, complete and consistent.

In summary this study found that MaramaAIC was able to enhance the quality of dialogue between a RE and client by showing the results of the captured and analysed requirements. The fast feedback and early validation by both parties contributed to better quality of the captured requirements.

Discussion

Our original research question was "***can automated support for Essential Use Case and Essential User Interface modelling enhance the consistency management and validation of requirements over manual methods?".*** We have answered this research question by developing MaramaAIC an automated toolset for EUC and EUI modelling and evaluating it via three quite different studies: we examined the tool's efficacy and performance in comparison to manual modelling approaches; the tool's usability and user perceived strengths and weaknesses for end-to-end rapid prototyping support; and finally the effectiveness of the tool in improving the dialogue between REs and clients to improve captured requirements quality.

Our studies showed positive results especially in terms of tool usefulness. They show a good degree of acceptance by end-users of the tool in automatically managing the consistency and validating requirements. Our results also appear to

complement prior studies in applying EUCs (Kamalrudin,Grundy & Hosking 2010), (Biddle etal., 2000). It was found by our subjects that our MaramaAIC provides better accuracy and takes lesser time than the manual extraction of EUC from the textual natural language requirements. It is able to detect many quality errors when the extracted EUC models are matched against the best practice EUC patterns. In this case, the detected errors are notified to the users using inconsistency warnings, problem markers and highlights. It was also demonstrated that our tool is able to assist both the RE and clients in the discussion, confirmation and validation of the captured requirements. Our tool is also able to ease and fasten the process of requirements validation via the end-to-end rapid protyping and the visualisation of effects that help to trigger fast feedback based on any impact of changes or modifications. As noted earlier, however, while encouraging these results can not be viewed as definitively answering our research question due to the limited number of test subjects (20 students and 6 professionals) and limited size and number of exemplar requirements used in the experiments.

However, there are some limitations on the functionality of the tool that requires enhancement. First, we found some problems when dealing with multiple requirements. Although the tool is able to support multiple requirement as described in the section tool usage example, the tool cannot perform a simultaneous traceback for both requirements. Secondly, it is able to perform trace back for one set of requirements at a time only. This somehow makes it difficult for the users to traceback the association of EUCs and EUIs model with the textual natural language requirements. Finally, the tool does not support partial selection of the change, although it provides highlights and an inconsistency warning for inconsistency detection that appear together with the options to either delete or cancel. Thus, this somehow affects the decision of validating the requirements.

Therefore, there are some improvements needed to improve the usability of MaramaAIC. We need to enhance layout to reduce the consistency issues noted and provide training material. The colour and shapes used in the tool need some improvement with better labelling to explain the features. The tool could also be integrated with a GUI template for the generated HTML form for each domain of application. Then, we need to improve the traceability support for multiple requirements where the tool should allow traceback for multiple requirements at the same time. Further, we also needs to consider to enhance the tool by supporting

partial selection on changes during the process of validation.The library for essential interaction patterns, EUC interaction patterns and EUI patterns also need expansion. To assist this, a pattern template editor needs to be developed to allow rapid authoring and update of the patterns to be done by any RE.

We believe our preliminary evaluation has shown that our end-to-end approach is a promising way of improving the dialogue between the REs and their clients. However, we need to conduct a longitudinal study to confirm this in extended practice. Other improvements include incorporating better NL processing support to complement our current abstract interaction extraction approach. In addition, we are exploring multi-lingual requirements capture and consistency management via EUCs built on our current end-to-end rapid prototyping approach (Kamalrudin,Grundy & Hosking 2012)

Summary

Inconsistency, Incorrectness and Incompleteness are common errors that always occur in requirements. Besides, there is also limited tool support that able to provide end-to-end support in validating and managing the consistency of requirements between the requirements engineers and their client. We have described an automated tool support called MaramaAIC using semi-formal models: Essential Use Cases (EUCs) and Essential User Interface (EUI) for managing requirements consistency and validation. This tool can automatically extract abstract interactions and EUC models from textual natural language requirements. Then, an EUI prototype model and concrete UI prototype can also be automatically generated from the EUC model. We have also demonstrated that these automation processes perform better than manual processes conducted by requirements engineers. In addition, our tool helps to automatically capture the essential requirements, check for the inconsistency, incorrectness and incompleteness using the developed essential interaction patterns and EUC interaction patterns with the traceability and visualisation support. Our tool is also able to automatically generate UI prototypes using the developed EUI patterns library, which helps to provide a clearer picture of the requirements to the client and help to ease the process to confirm the consistency of the requirements captured by the requirements engineers against the client's original requirements.

Acknowledgement

We acknowledge the support of the participants in our evaluation studies who willingly gave their time. Massila Kamalrudin acknowledges financial support from the University of Auckland, Swinburne University of Technology, Ministry of Higher Education Malaysia (FRGS/F00185) and Universiti Teknikal Malaysia Melaka (UTeM) for their assistance in this research. All authors acknowledge the support of the New Zealand Ministry of Business, Innovation & Employment via funding for the Software Process and Product Improvement project. We also thank Jun Huh for his assistance in developing MaramaAIC and Mark Young for his kindness in providing us the exemplar requirements. Finally, we thank the extremely thorough and detailed comments of the anonymous referees who went above and beyond the call of duty to give us very precise, detailed and very helpful assistance on earlier drafts of this article.

References

Am, Sampaio, R., Chitchyan, R., Rashid, A. & Rayson, P.: EA-Miner: a tool for automating aspect-oriented requirements identification. Paper presented at the Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, Long Beach, CA, USA (2005)

Ambler, S. W.:Essential (Low Fidelity) User Interface Prototypes. http://www.agilemodeling.com/artifacts/essentialUI.htm (2003-2009). Accessed 20 April 2010

Ambler, S. W.: The Object Primer: Agile Model-Driven Development with UML 2.0 (3rd ed.), New York Cambridge University Press (2004)

Bjork, R. C.: Use Cases for Example ATM System. http://www.math-cs.gordon.edu/courses/cs320/ATM_Example/UseCases.html (June 1998). Accessed February 2009

Biddle, R., Noble, J. & Tempero, E.: Essential use cases and responsibility in object-oriented development. *Aust. Comput. Sci. Commun., 24*(1), pp. 7-16, (2002)

Biddle, R., Noble, J. & Tempero.E.: Pattern for Essential Use Cases (C. science, Trans.) (Vol. CS-TR-01/02). Wellington, New zealand: Victoria University of Wellington (April 2000)

Blackwell, A., Britton, C., Cox, A., Green, T., Gurr, C., Kadoda, G. & Young, R.: Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In M. Beynon, C. Nehaniv & K. Dautenhahn (Eds.), Cognitive Technology: Instruments of Mind, vol. 2117, pp. 325-341. Springer Berlin / Heidelberg (2001)

Blackwell, T. G. a. A.: Cognitive Dimensions of Information Artefacts:a tutorial. Version 1.2. https://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf (1998)

Buskirk, V. R. & Moroney, B. W.: Extending prototyping. *IBM Systems Journal, 42*(4), pp. 613-623., (2003)

Constantine, L. L.: Rapid Abstract Prototyping Software development 6 (11), 1998

Constantine, L. L., & Lockwood, L. A. D. : Software for use: a practical guide to the models and methods of usage-centered design: ACM Press/Addison-Wesley Publishing Co., (1999)

Corporation, B. S.: CaliberRM™ Enterprise Software Requirements Management System. http://www.borland.com/us/products/caliber/index.html (2011). Accessed 08 February 2011

Cristian, B.: Generating an Abstract User Interface from a Discourse Model Inspired by Human Communication, (2008)

Dardenne, A., Van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. Science of computer programming, 20(1), 3-50.

Denger, C., Berry, D. M. & Kamsties, E.: Higher Quality Requirements Specifications through Natural Language Patterns. Paper presented at the Proceedings of the IEEE International Conference on Software-Science, Technology \& Engineering, pp. 80.80-7695-2047-7692: IEEE Computer Society (2003)

Egyed, A.: Scalable Consistency Checking Between Diagrams-The ViewIntegra Approach. Proceedings of the 16th IEEE international conference on Automated software engineering, pp. 387. IEEE Computer Society, (2001)

Evans, G.: Getting from use cases to code, Part 1: Use-Case Analysis. http://www.ibm.com/developerworks/rational/library/5383.html. Accessed January 2009

Fabbrini, F., Fusani, M., Gnesi, S. & Lami, G.: The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. Paper presented at the Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard, 2001

Finkelstein, A., & Emmerich, W.: The future of requirements management tools. Information Systems in Public Administration and Law, (2000)

Geisser, M., Hildenbrand, T. & Riegel, N.: Evaluating the Applicability of Requirements Engineering Tools for Distributed Software Development (D. o. I. S. 1, Trans.) *Working Paper 2/2007* (Working Papers in Information Systems ed.). Germany: University of Mannheim., 2007

Gervasi, V. & Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Trans. Softw. Eng. Methodol., 14(3), pp. 277-330. , 2005

Glinz, M.: A lightweight approach to consistency of scenarios and class models, Proc.4th International Conference on Requirements Engineering 2000, 2000, pp. 49-58., (2000)

Grundy, J. C., Hosking, Huh, J. & Li, N.: Marama: an Eclipse meta-toolset for generating multi-view environments. Paper presented at the 2008 IEEE/ACM International Conference on Software Engineering, Liepzig, Germany, May 2008

Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W. & Schwinger, W.: Automated verification of model transformations based on visual contracts. Automated Software Engineering, 1-42. doi: 10.1007/s10515-012-0102-y

Horton, T.: Example Use Cases for PARTS. http://www.cs.virginia.edu/~horton/cs494/examples/parts/usecases-ex1.html. Accessed February 2009

Hull, E., Jackson, K. & Dick, J.: DOORS: A Tool to Manage Requirements Requirements Engineering, pp. 173-189. Springer, London (2005)

Huzar, Z., Kuzniarz, L., Reggio, G. & Sourrouille, J. L.: Consistency Problems in UML-Based Software Development *UML Modeling Languages and Applications,* pp. 1-12., (2005)

IBM. Rational RequisitePro A requirements management tool. http://www-01.ibm.com/software/awdtools/reqpro/. Accessed 13 February 2011

Inc., S. S.: Serena. Requirements Management The Proven Way to Accelerate Development. http://www.serena.com/docs/repository/products/rm/wp900-001-0505.pdf (2011). Accessed 14 February 2011

Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. (1999). The unified software development process (Vol. 1). Reading: Addison-wesley.

Kim, J., Park, S. & Sugumaran, V. : Improving use case driven analysis using goal and scenario authoring: A linguistics-based approach, Data & Knowledge Engineering, vol. 58, pp. 21-46, (2006)

Kamalrudin, M., Grundy, J. & Hosking, J.,: Tool Support for Essential Use Cases to Better Capture Software Requirements. Paper presented at the 25th IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, 20-24 September 2010

Kamalrudin, M. & Grundy, J.: Generating essential user interface prototypes to validate requirements. Paper presented at the Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, 2011

Kamalrudin, M., Ahmad, S.S., Sidek, S. & Daud, N.: A Review of Requirements Engineering Tools for Requirements Validation Software Engineering Process, International Journal of Software Engineering, IJSET, vol. 1, (2014)

Kamalrudin, M., Grundy, J. & Hosking, J.: MaramaAI: Tool Support for Capturing and Managing Consistency of Multi-lingual Requirements, 27th Automated software Engineering Conference, Essen, Germany, (2012)

Kotonya, G. & Sommerville, I.: Requirement Engineering Process and Techniques. West Sussex, England: John Wiley & Sons Ltd, (1998)

Kovacevic, S. UML and User Interface Modeling The Unified Modeling Language. «UML»'98: Beyond the Notation, pp. 514-514., (1999)

Kutar, M., Britton, C. & Wilson, J.: Cognitive Dimensions An Experience Report. Paper presented at the Twelfth Annual Meeting ofthe Psychology of Programming Interest Group, Memoria, Cozenza Italy, (2000)

Lund, A.: USE Questionnaire Resource Page. http://usesurvey.com/IntroductionToUse.html (2009). Accessed February 2010

Lamport, L. (2002). Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co., Inc..

Lang, M. & Duggan, J.: A Tool to Support Collaborative Software Requirements Management. Requirements Engineering, 6(3), 161-172(2001). doi: 10.1007/s007660170002

Larry, L. C. & Lucy, A. D. L.: Structure and style in use cases for user interface design Object modeling and user interface design: designing interactive systems. pp. 245-279. Addison-Wesley Longman Publishing Co., Inc., (2001)

Larry, L. C. & Lucy, A. D. L.:Usage-centered software engineering: an agile approach to integrating users, user interfaces, and usability into software engineering practice. Paper presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, (2003)

Lopez-Herrejon, R. E. & Egyed, A.: Towards fixing inconsistencies in models with variability. Paper presented at the Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, Leipzig, Germany, (2012)

Ltd, S. D.: Creative New Media. http://www.silicon-dream.com/(1996-2011). Accessed 25 May 2010

Memmel, T., & Reiterer, H.: Inspector: Interactive UI Specification Tool *Computer-Aided Design of User Interfaces VI*. pp. 163-175., (2009)

Neill, C. J. & Laplante, P. A.: Requirements engineering: the state of the practice. *Software, IEEE, 20*(6), pp. 40-45., (2003)

Nentwich, C., Wolfgang, E. & Anthony, F.: Consistency management with repair actions. Paper presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, (2003)

Nguyen, T. H., Vo, B. Q., Lumpe, M. & Grundy, J.: *REInDetector: a framework for knowledge-based requirements engineering*. Paper presented at the Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, Essen, Germany, (2012)

Nuseibeh, B., Easterbrook, S. & Russo, A.: Leveraging Inconsistency in Software Development. *Computer, 33*(4), pp. 24-29., (2000)

Perrouin, G., Brottier, E., Baudry, B. & Le Traon, Y.: Composing Models for Detecting Inconsistencies: A Requirements Engineering Perspective *Requirements Engineering: Foundation for Software Quality* , pp. 89-103)., (2009)

Reder, A. & Egyed, A.: Incremental Consistency Checking for Complex Design Rules and Larger Model Changes. In R. France, J. Kazmeier, R. Breu & C. Atkinson (Eds.), Model Driven Engineering Languages and Systems. vol. 7590, pp. 202-218. Springer, Berlin Heidelberg (2012)

Robertson, S. & Robertson, J.: Mastering the Requirements Process (2nd Edition): Addison-Wesley Professional, (2006)

Sardinha, A., Chitchyan, R., Weston, N., Greenwood, P. & Rashid, A. EA-Analyzer: automating conflict detection in a large set of textual aspect-oriented requirements. Automated Software Engineering, pp. 1-25. doi: 10.1007/s10515-012-0106-7

Satyajit, A., Hrushikesha, M. & George, C.: Domain consistency in requirements specification Quality Software, 2005. (QSIC 2005). Fifth International Conference on. pp. 231-238. 1550-6002., (2005)

Scenario examples. http://www.opensrs.com/resources/documentation/sync/scenarioexamples.htm. Accessed February 2009

Some, S. S.: *Use Cases based Requirements Validation with Scenarios*. Paper presented at the Proceedings 13th IEEE International Conference in Requirements Engineering 2005, (2005)

Tjong, S. F., Hallam, N. & Hartley, M. : Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns. Paper presented at the Computer and Information Technology, 2006. CIT '06. The Sixth IEEE International Conference, (2006)

Yijun, Y.:From Goals to Aspects: Discovering Aspects from Requirements Goal Models. In Proc. 12th IEEE International Requirements Engineering Conference 2004, (2004), 6-11 Sept. 2004.

Yu, E. S. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on (pp. 226-235). IEEE.

Yue,T., Briand, L.C., Labiche,Y.: aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models. ACM Trans. Softw. Eng. Methodol. 24(3), 13 (2015)

Yufei, X., Tao, T., Tianhua, X. & Lin, Z.: Research on requirement management for complex systems. Paper presented at the 2nd International Conference Computer Engineering and Technology (ICCET), 2010, 16-18 April 2010, (2010)

Zisman, G. S. a. A.: Handbook of Software Engineering and Knowledge Engineering. In S. K. Chang (Ed.), (Vol. Volume 1, pp. 329-380): World Publishing co., (2001)

Zhang, G., Matthias M. Hölzl: Weaving semantic aspects in HiLA. AOSD 2012, 263-274(2012)

Zhang, G.: Aspect-Oriented Modeling of Mutual Exclusion in UML State Machines. ECMFA 2012, 162-177(2012)