An Automated Approach to Embrace Changes During Use case Model Evolution

Dr. Amer AbuAli
Department of Software Engineering,
Faculty of Information Technology
P.O. Box 1 Philadelphia University, 19392, Amman, Jordan.
Tel: 00 962 6 4799000

Abstract:
Use case model is subject to changes throughout the software development life cycle. Impacts of these changes affect directly the requirements and consequently the resulted system. Scrapping and replacing use case is expensive; in this paper we proposed a solution that integrates changes in use case in requirement phase. This solution combines independent enhancements to some version of a use case into a new version that include the enhancements and the old use case. CASE tool implementation and experimental evaluation of the proposed approach showed promising results in terms of software development time saving and better use case models integrity.

Keywords: Requirement engineering, Functional requirements, Use case changes, Use case evolution.

1. Introduction

Understanding the requirements of a problem is among the most difficult tasks that face a software engineer. Requirement engineering (RE) helps software engineers to better understand the problem they will work to solve. It encompasses the set of tasks that lead to an understanding of what the business impact of the software will be, what the customer wants, and how end-users will interact with the software [1, 2].
Most of the changes into software can be traced back to the early requirements stage when a recovery action can still be cost-effective [3]. Such changes may become necessary because of changes in the real-world context in which the proposed system would be situated or because of changes in stakeholder perceptions of the proposed system. Requirements Evolution involves updating a description of user requirements for a target system to accommodate new requirements or to remove existing ones [4, 5].
To capture functional requirements, that are statements of the services that the system must provide or are descriptions of how some computations must be carried out [6, 7], the widespread practice is the use case model. It describes the functional requirements of a software system and is used as input to several activities in a software development project. It gives a high-level view of the requirements of a system. The quality of the use case model therefore has an important impact on the quality of software [8].
Use case model is subject to changes sometimes later in software life cycle. Changes are due to 1) market demands, such as a large customer wanting things done their way; 2) business requirement change, such as new policies or operational processes; 3) legislative and regulatory change; and 4) imaginative users. Impacts of these changes affect directly the requirements and consequently the product [3, 9].
Here, we faced two problems: (1) scrapping and replacing use cases or (2) merging changes in order to create new use case. The former is more expensive, we propose an original solution to the second problem.
Use case merging is essential to deal with parallel modifications carried out by different requirement engineers that are not necessarily aware of each other's changes. Our solution combines various independent enhancements of a given version of a use case into a new use case that includes the semantics of both the enhancements and the old use case. In this context, changes are brought in separate copies of the old use case. Copies as well as the old use case are compared and merged in order to produce a new version including all modifications. This approach provides computer aid for combining the results of several people's separate efforts. This approach is inspired from our previous researches in software merging where we have proposed a new approach for program integration [10, 11].
The outline of this paper is as follows: Section 2 presents a background about the use case concept. Section 3 discusses our approach to use case modelling. Section 4 illustrates our identification of changes by an example. Section 5 shows the manner of merging use cases. Sections 6 and 7 demonstrate the tool support and experimental use of the proposed approach. Finally, Section 8 concludes our research direction.

2. Background

Employment of use cases is now common practice in software development, and use case is now a recognized concept in development processes [12, 13].

A use case is an object-oriented modeling construct that is used to define the behavior of a system. Interactions between the user and the system are described through a prototypical course of actions along with a possible set of alternative courses of action. Primarily, use cases have been associated with requirements gathering and domain analysis. However, with the release of the Unified Modeling Language (UML) specification version 1.5 [14], the scope of use cases has broadened to include modeling constructs at all levels. Due to this expanded scope, the representation of use cases has taken on increasing importance.

A *use case* defines a goal-oriented set of interactions between external actors and the system under consideration. *Actors* are parties outside the system that interact with the system [14]. An actor may be a class of users, roles users can play, or other systems. A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It also includes possible variants of this sequence, e.g., alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure to complete the service because of exceptional behavior, error handling, etc. The system is treated as a "black box", and the interactions with system, including system responses, are as perceived from outside the system [12, 13].

According to UML version 1.5 [14] we describe, briefly, the types of relationships of use case as below:

i) Actor relationships

There is one standard relationship among actors and one between actors and use cases, called generalization and association respectively. A generalization from an actor A to an actor B indicates that an instance of A can communicate with the same kinds of use-case instances as an instance of B. Association is related to the participation of an actor in a use case, i.e. instances of the actor and instances of the use case communicate with each other.

ii) Use case relationships

In addition to the association, described previously, there are several standard relationships among use cases or between actors and use cases. A generalization from use case A to use case B indicates that A is a specialization of B. An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented (subject to specific conditions specified in the extension) by the behavior specified by A. The behavior is inserted at the location defined by the extension point in B which is referenced by the extend relationship. While an include relationship from use case A to use case B indicates that an instance of the use case A will also contain the behavior as specified by B. The behavior is included at the location which defined in A.

3. Use case modelling

To permit an automatic use case analysis which is implicit in the conventional representation, an explicit representation needs an internal form.

3.1 Internal form

In the context of use case understanding and modification (evolution), a dependence relationship of a use case model is defined formally by the 5-tuples:

<As, At, Rel, Typ, Id>.

It means that target actor/use case **At** depends on actor/use **As** according to the relationship **Rel** with the type **Typ** for the relationship **Id**.

Rel is a relationship that can be a generalization between actors/use cases (**Gen**), an association between actor and use case (**Ass**), an extend (**Ext**), or an include between use cases (**Inc**).

Typ is dedicated to the type of multiplicity in a given association (n..m), it is the number of possible instances of actors associated with a single instance of use case.

Id is a unique identifier corresponding to relationship number**.**

3.2 Modeling

3.2.1 Modeling actor relationships

An association is formalized by the following 5-tuplet: <Actor, Use case, Ass, Mul, Id>.

It means that instances of **Actor** and instances of **Use case** communicate with a multiplicity **Mul** in the association **Ass** numbered **Id**.

A generalization between actors is represented by: <Actor1, Actor2, Gen, ϕ, Id>

It expresses that **Actor2** inherits (Gen) from **Actor1** in the relationship numbered **Id**.

3.2.2 Modeling use case relationships

We express a generalization between use cases by: <Use case1, Use case2, Gen, ϕ, Id>

It means that **Use case2** inherits (Gen) from **Use case1** in the relationship numbered **Id**.

An include relationship is expressed by: < Use case1, Use case2, Inc, ϕ, Id>

It indicates that an instance of the **Use case1** will also contain the behavior as specified by **Use case2** in the relationship numbered **Id**.

An extend relationship is expressed by: <Use case1, Use case2, Ext, ϕ, Id>

It means that an instance of **Use case2** may be augmented (**Ext**) by the behavior specified by **Use case1** in the relationship numbered **Id**.

In order to illustrate our approach Figure 1 presents the use case of an ordering system and the corresponding internal form (Table1).

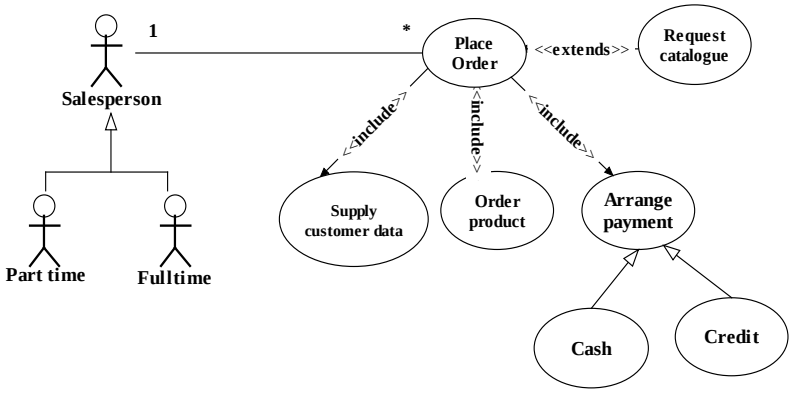

Fig. 1. Use case of an ordering system and its internal form of use case.

< Salesperson, Part time, Gen, ϕ, 1>,
< Salesperson, Full time, Gen, ϕ, 2>,
< Salesperson, Place Order, Ass, (1, *), 3>,
<Place Order, Supply Customer Data, Inc, ϕ, 4>,
<Place Order, Order Product, Inc, ϕ, 5>,
<Place Order, Arrange Payment, Inc, ϕ, 6>,
<Place Order, Request Catalogue, Ext, ϕ, 7>,
<Cash, Arrange Payment, Gen, ϕ, 8>,
<Credit, Arrange Payment, Gen, ϕ, 9>.

Table 1. Internal form of ordering system use case

4. Identification of changes

Use case changes can be syntactic or semantic. Syntactic changes concern changes of actors, use cases, and relationship names. Semantic changes concern semantic changes of actors, use cases, and relationships.

Semantic changes of actors/use cases can be adding/deleting actors/use cases. Semantic changes of relationships not only occur with previous changes but also with redirecting edges or changing type of relationships.

In order to illustrate this approach, we propose to apply it in the following example. According to use case model **Base** of figure 1, two variants are proposed. In variant A (figure 2), we add a new actor "Trainee", change the multiplicity (1..*) by (1..5), and change "Salesperson" by "Salesperson Team". In other words we make two semantic changes and one syntactic change, namely adding new actor changing the multiplicity and renaming "Salesperson" by "Salesperson Team" Table 2 gives the internal form of variant A. In variant B (figure 3), we add a new use case "Log in", redirect use case "Request catalogue" to actors and make syntactic change of "cash" and "credit" by "cash payment" and "credit payment". Table 3 gives the internal form of variant B.

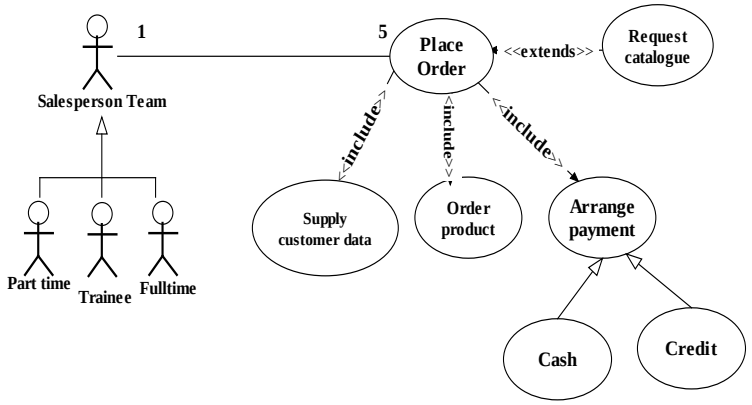

Fig. 2. Variant A of use case Base**.**

< Salesperson Team, Part time, Gen, ϕ, 1>,
< Salesperson Team, Full time, Gen, ϕ, 2>,
< Salesperson, Place Order, Ass, (1, 5), 3>,
<Place Order, Supply Customer Data, Inc, ϕ, 4>,
<Place Order, Order Product, Inc, ϕ, 5>,
<Place Order, Arrange Payment, Inc, ϕ, 6>,
<Place Order, Request Catalogue, Ext, ϕ, 7>,
<Cash, Arrange Payment, Gen, ϕ, 8>,
<Credit, Arrange Payment, Gen, ϕ, 9>
< Salesperson Team, Trainee, Gen, ϕ, 10>.

Table 2. Internal form of Variant A.

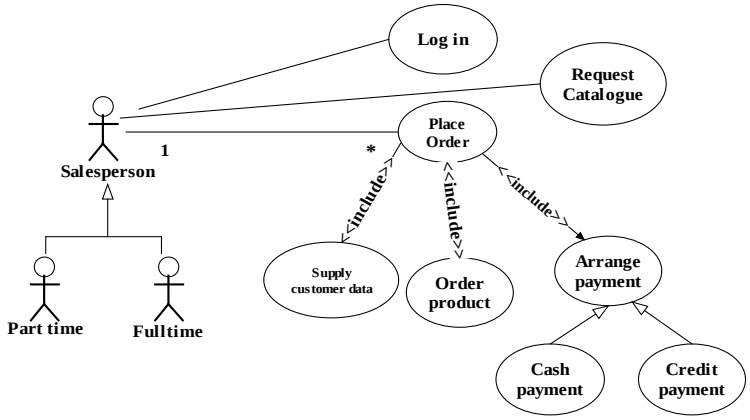

Fig. 3. Variant B of use case Base

< Salesperson, Part time, Gen, ϕ, 1>,
< Salesperson, Full time, Gen, ϕ, 2>,
< Salesperson, Place Order, Ass, (1, *), 3>,
<Place Order, Supply Customer Data, Inc, ϕ, 4>,
<Place Order, Order Product, Inc, ϕ, 5>,
<Place Order, Arrange Payment, Inc, ϕ, 6>,
<Cash Payment, Arrange Payment, Gen, ϕ, 8>,
<Credit Payment, Arrange Payment, Gen, ϕ, 9>
< Salesperson, Log in, Ass, (ϕ), 10>,
< Salesperson, Request Catalogue, Ass, (ϕ), 11>.

Table 3. Internal form of Variant B.

4.1. Actors changes

Actor changes concern the change of name (syntactic) or the behavior (semantic) of a given actor. Semantic changes can be adding, deleting actors, and/or redirecting the relationships from these actors. By comparing actors of each variant according to actors of use case Base, we can identify actor changes. Changes are grouped in four sets: UA, ACN, ACB, and ACNB.

UA set contains Unaltered Actors in all use cases. This concerns actors keeping the same name and the same behavior in all variants. Informally, it is interpreted by the same internal form (5-tuples) of actors all in variants.
Let A= (A_{11}, A_{12},…, A_{ij}, …, A_{nm}) to denote actor i in the use case model j
UA = { A_{ij} / <A_{ij}, A_{kj}, Rel_{ij}, Typ_{ij}, Id_{ij} >$_{Base}$ = <$A_{i'j'}$, $A_{k'j'}$, $Rel_{i'j'}$, $Typ_{i'j'}$, $Id_{i'j'}$>$_{variant}$ }
In our example, this set is concerned by the following actors: "Full Time" and "Part Time".

ACN is the set of Actors with Changed Names, but keeping the same behavior. Informally, it is interpreted by changing only the name of actor in the specific 5-tuples.

$ACN = \{ A_{ij} / <A_{ij}, A_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} = <A_{i'j'}, A_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \wedge "A"_{ij} \neq "A"_{i'j'}\}$
In our example, ACN is concerned by actors: Salesperson replaced by Salesperson Team in variant A.

ACB is the set of Actors with Changed Behaviors but keeping the same names. As stated previously this concerns redirecting relationships or changing the relationship types.
$ACB = \{\forall A_{ij} / <A_{ij}, A_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} \neq <A_{i'j'}, A_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \wedge " A"_{ij} = "A"_{i'j'}"\}$
In variant A, ACB is concerned by a new inheritance between "Salesperson" and the new actor "Trainee", while in variant B we have added two associations (with "Request catalogue" and "Log in").

ACNB (Actors with Changed Names and Behavior) set is concerned by adding/deleting actors.
$ACNB = \{\forall A_{ij} / <A_{ij}, A_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} \neq <A_{i'j'}, A_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \}$
In our example, ACB is concerned by adding a new actor "Trainee".
We note that A_{kl} and $A_{k'j'}$ can be actor or use case.

4.2. Use cases changes
Use case changes is concerned by syntactic change or semantic of a given use case. Semantic changes can be adding, deleting use cases, and/or redirecting the relationships from these use cases. By comparing use cases of each variant according to use cases of Base, we can identify use cases changes. Changes are grouped in four sets: UUC, UUCN, UUCB, and UUCNB.
UUC set contains Unaltered Use Cases. This concerns use cases keeping the same name and the same behavior in all variants. Informally, it is interpreted by the same internal form (5-tuples) of this use case in variants.
Let $UU= (UU_{11}, UU_{12},\ldots, UU_{ij}, \ldots, UU_{nm})$ to denote use case i in the use case model j
$UUC = \{U_{ij} / <U_{ij}, U_{kj}, Rel_{ij}, Typ_{ij}, Id_{ij}>_{Base} = <U_{i'j'}, U_{k'j'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \}$
In our example, this set is concerned by the following use cases: "Place Order", "Supply customer data", "Order Product", and "Arrange payment".

UUCN is the set of Use Cases with Changed Names, but keeping the same behavior. Informally, it is interpreted by changing only the name of actor in the specific 5-tuples.
$UUCN = \{U_{ij} / <U_{ij}, U_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} = <U_{i'j'}, U_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \wedge "U"_{ij} \neq "U"_{ij}\}$
In our example, UUCN is concerned by use cases Cash and Credit replaced by Cash payment and Credit payment in variant B.

UUCB is the set of use cases with Changed Behaviors but keeping the same names, this concerns redirecting relationships or changing the relationship types.

$UUCB = \{ U_{ij} / <U_{ij}, U_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij}>_{Base} \neq <U_{i'j'}, U_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \wedge " "U_{ij}"= "U_{ij}"\}$.

In variant B, use case "Request Catalogue" is redirected to "Salesperson" instead of "Place Order" and the type of relationship (<<include>>) is changed into a normal association.

UUCNB (Use Cases with Changed Names and Behavior) set is concerned by adding/deleting use cases.
$UUCNB = \{U_{ij} / <U_{ij}, U_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} \neq <U_{i'j'}, U_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant}\}$
In our example, UUCNB is concerned by adding a new use case: "Log in" in variant B.
We note that U_{kl} and $U_{k'j'}$ can be actor or use case.

4.3 Relationships changes

Also relationship changes concern the syntactic change or semantic of a given relationship. Semantic changes can be adding, deleting, and/or redirecting relationships. By comparing relationships of each variant according to relationships of use case Base, we can identify relationship changes. Changes are grouped in four sets: UR, RCN, RCB, and RCNB.

UR set contains Unaltered Relationships in all use cases. This concerns actors keeping the same name and the same behavior in all variants. Informally, it is interpreted by the same internal form (5-tuples) of this relationship in variants.
Let $R= (R_{11}, R_{12},\ldots, R_{ij}, \ldots, R_{nm})$ to denote relationship i in the use case model j.
$UR= \{R_{ij} / <A_{ij}, A_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} = <A_{i'j'}, A_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant}\}$

In the example, this set is concerned by an inheritance from "Part time" and "Full time" to "Salesperson", <<include>> associations from "Supply customer data", "Order Product", and "Arrange payment" to "Place Order", and an inheritance from "Cash" and "Credit" to "Arrange payment".

RCN is the set of Relationships with Changed Names, but keeping the same behavior. Informally, it is interpreted by changing only the name of relationship in the specific 5-tuples.
$RCN = \{A_{ij} / <A_{ij}, A_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} = <A_{i'j'}, A_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \wedge "Rel"_{ij} \neq "Rel"_{i'j'}\}$

RCB is the set of Relationships with Changed Behaviors but keeping the same names. As stated previously this concerns redirecting relationships or changing the relationship types.
$RCB = \{A_{ij} / <A_{ij}, A_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij} >_{Base} \neq <A_{i'j'}, A_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant} \wedge "Rel"_{ij} = "Rel"_{I'j'}\}$

In variant A there is a change of multiplicity with use case "Place Order" (1..5 instead of 1..*). In varian

we have three new relationships: two associations with use cases "Log in" and "Request catalogue", and an inheritance with a new actor "Trainee".

RCNB (Relationships with Changed Names and Behavior) set is concerned by adding/deleting relationships.
RCNB = $\{A_{ij} / <A_{ij}, A_{kl}, Rel_{ij}, Typ_{ij}, Id_{ij}>_{Base} \neq <A_{i'j'}, A_{k'l'}, Rel_{i'j'}, Typ_{i'j'}, Id_{i'j'}>_{variant}\}$
In our example, RCNB is concerned by (1) adding new association from "Log in" to "Salesperson" redirecting and changing the <<include>> association between "Request Catalogue" and "Salesperson" into a normal association.

5. Generation of the new version of use case

We generate the new version of use case according changes identified previously. Unaltered actors, use cases, and relationships sets (UA, UUC, and UR) are kept. Actors, use cases, and relationships with changed names in variants replace corresponding actors, use cases, and relationships of Base (from ACN, UUCN, and RCN). Actors, use cases, and relationships with changed behavior of variants replace corresponding actors, use cases, and relationships of Base (from ACB, UUCB, and RCB). Actors, use cases, and relationships with changed names and behaviors of variants, interpreted by insertions or deletions, are inserted or deleted (from ACNB, UUCNB, and RCNB). Finally we obtain an internal form corresponding to the new use case (Table 4).
However there is a possible way in which we can fail to represent a satisfactory merged use case model. In Software merging [15, 16] these are referred as "Type I and Type II interference". Type I occurs when we make the same changes to the same actor, use case, relationship or multiplicity in different variants. In this case what is the change handled in the new version? Type II interference occurs when reconstituting the merged use case diagram from the internal form, it can be an infeasible graph.
If there are no interferences we can reconstitute the new use case diagram. Figure 4 illustrates a reconstitution of use case diagram from the internal form of Table 4.

< Salesperson, Part time, Gen, ϕ, 1>,
< Salesperson, Full time, Gen, ϕ, 2>,
< Salesperson, Place Order, Ass, (1, *), 3>,
<Place Order, Supply Customer Data, Inc, ϕ, 4>,
<Place Order, Order Product, Inc, ϕ, 5>,
<Place Order, Arrange Payment, Inc, ϕ, 6>,
<Cash Payment, Arrange Payment, Gen, ϕ, 8>,
<Credit Payment, Arrange Payment, Gen, ϕ, 9>
< Salesperson Team, Trainee, Gen, ϕ, 10>,
< Salesperson, Log in, Ass, (ϕ), 11>,
< Salesperson, Request Catalogue, Ass, (ϕ), 12>.

Table 4. Internal form of the new version of use case

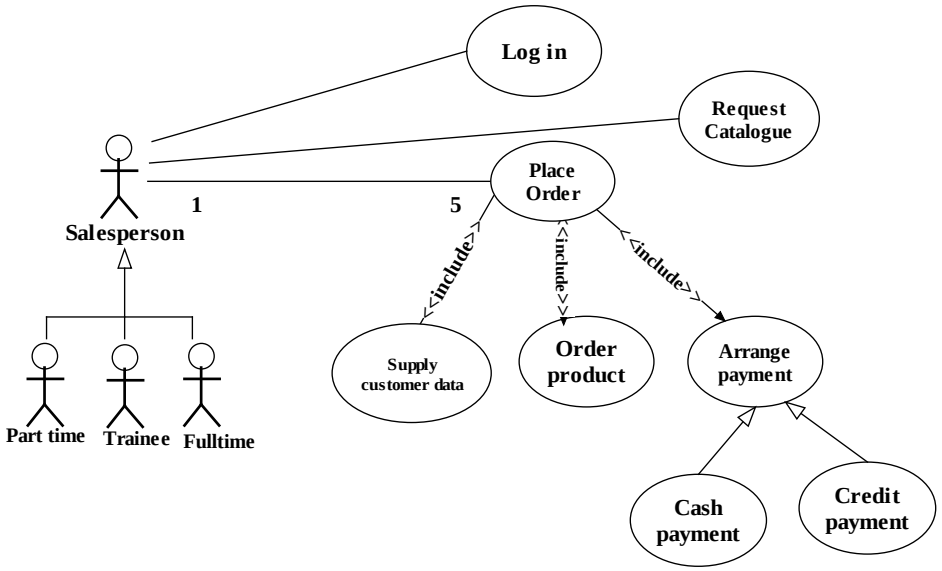

Fig. 4. The new version of use case diagram

6. Automation of Proposed Approach

The automated support of the proposed approach passed through a number of rationales. Examples include: what type of automation should be supported?, what development approach should be adopted?
The main debate that faced the research team is related to the software development approach. Two options were available: 1) develop limited capabilities stand alone CASE tool, and 2) develop an integrated shell for an already existing CASE tool. The 4+1 architectural views [17] suggest that any system has five views: design, implementation, process, deployment and use case. Activities within a view require information from other views. Elements from one view depend on or be driven by those of another. Moreover, the views may need to be ordered so that the information shared between two or more views remains consistent. An exception to this rule occurs with the use case view which is defined to drive the development of other system views. As the main output of the proposed approach is the use case model of the anticipated system and being a core model in software development, it was decided to go with the second development approach to support development of other system models using facilities of underlying CASE tool.

A survey on available CASE tools identified a number of commercial [18,19,20] and open source [21,22,23] CASE tools. Commercial tools (e.g. Rational Rose) ruled out of the candidate tools list due to expensive licensing cost which will inhibit accessibility of our approach to large number of users who are unable or unwilling to pay licensing cost. Therefore, three open source tools were short listed: StarUML [21], ArgoUML [22], and Netbeans Plug-ins [23]. Table 5 compares the features of the three tools. StarUML, as can be concluded from table 5, supersedes the other two tools in a number of factors. Hence, it was selected as a platform for the automation of the proposed approach

.

Table 5. CASE Tools Comparisons Summary**.**

Tool	UML Supported Version	Help and User Support Available	All Diagrams Supported	Portable?	Maintainable, Usable, and Extensible?	Support to Recent Trends in Software Modelling (e.g. MDA, NX)
StarUML	2.0	Yes	Yes	Yes	High	Yes
ArgoUML	1.4	No	Yes	Yes	Med.	No
Netbeans	1.4	No	No	No	Low	No

7. The Proposed Approach in Operation

TestWarehouse is a medium size software house. The main unit of software development projects is a team. Each team consists of up to 18 resources of different roles: project manager(s), IT technical support officer(s), system and business analysts, developers, and software quality engineers.

The adopted software development process in TestWarehouse projects differs from one project to another according to project context including project type, technical experience, application domain, delivery constraints, resources, and surrounding risks. However, the software development processes recently used in TestWarehouse are: eXtreme Programming, Scrum, and Rational Unified Process (RUP). These software process models are use case based and embrace frequent requirements changes which make them good test bed for the proposed use case evolution approach.

The proposed approach has been in operation for 8 months and utilized by TestWarehouse's business and system analysts in six projects. Table 6 demonstrates projects demographics in relation to project size, type, application domain, number of use cases, and number of requirements changes.

The main reported advantage of using the catalogue was the noticeable time saving in requirement engineering phase. This is attributed to reusability of use cases. Reported time saving percentages varied between 8% and 25% of the total software development project time. Analyzing the reasons behind the high fluctuation in reported time saving percentages, it was found that this is attributed to a number of factors including: (1) number of use cases in the project, and (2) use case complexity. In addition, users reported that models generated using the

proposed approach possess better completeness and comprehensiveness characteristics.

Table 6. Demographics of Experimental Projects**.**

Project	Size	Type	Application Domain	Number of Use Cases	Number of Requirements Changes
1	Small	In house development	Human Resources	20	10
2	Medium	Product	Financial	45	18
3	Medium	Custom Development	e-Commerce	42	21
4	Medium	Outsourcing	Financial	50	20
5	Large	Support Project	CRM Software	80	35
6	Large	Custom Development	Insurance	71	40

8. Conclusion and future work

Use case model is subject to changes sometimes later in software life cycle. Impacts of these changes affect directly the requirements and consequently the resulted system. Scrapping and replacing use case is expensive; in this paper we have proposed an original solution to integrate changes in old use case in requirement phase.

This solution is based on (1) an internal form to represent formally dependencies between concepts of use cases, (2) identification of changes from this internal form, and (3) merging old use cases diagrams in order to obtain a new version that takes account all modifications if there is no conflict.

The proposed approach has been implemented on top of an open source CASE tool. Actual experimental work of the automated proposed approach showed its ability to save up to 25% of software development time with better completeness and comprehensiveness characteristics.

As a future work, we plan to incorporate this technique of modification to the next diagrams of Object Oriented Analysis and Design (interaction diagrams, state diagrams, activity Diagrams, etc.). In addition, further testing using further projects and users is planned to take place for this approach.

9. References

1. Nguyen L, Swatman PA (2003) Managing Requirement Engineering Process. Requirement Engineering Journal Volume 8 No 1: pp. 55-68.
2. Lamsweerde VA (2000) Requirements engineering in the year 00: A research perspective. ICSE'2000 pp. 519 Ireland.
3. Rolland C, Salinesi C, Etien A (2004) Eliciting gaps in Requirements Change. Requirement Engineering Journal Volume 9 Number 1 pp. 1-15.
4. Anderson S, Felici M (2000) Requirements changes risk/cost analyses: An avionics case study. SRA-EUROPE Annual Conference, Volume 2 pp. 921-925 Scotland.
5. PROTEUS Project (1996) Meeting the challenge of changing requirements. Deliverable 1.3, Centre for Software Reliability, University of Newcastle.
6. Ghose A (1999) Managing Requirements Evolution: Formal Support for Functional and Non-functional Requirement. IWPSE'1999 pp. 118-124 Japan.
7. Anderson S, Felici M (2001) Requirements evolution: From process to product oriented management. PROFES'2001.
8. Lascio L (2002) Towards an inspection technique for use case models. SEKE '02 pp. 127-134.
9. Zhang L, Xie D, Zou W (2001) Viewing use cases as active objects. ACM SIGSOFT Software Engineering Notes**,** Volume 26 Issue 2.
10. Khammaci T, Bouras Z.E (2002) Versions of program integration. In World Scientific (eds). Handbook of Software Engineering and Knowledge Engineering No 2, Pittsburg (USA), pp. 495-516. 2002.
11. Bouras Z.E, Khammaci T, Ghoul S (2000) A new approach for program Integration. The South African Computer Journal, No. 25, pp.3-11.
12. Cockburn A (1997) Structuring Use Cases with Goals. Journal of Object-Oriented Programming, Sep-Oct 1997.
13. Biddle R, Noble J, Tempero E (2002) Essential use cases and responsibility in object-oriented development. CRPITS'02 Volume 24 Issue 1.
14. Unified Modeling Language, Version 1.5, http://www.rational.com/uml
15. Horwitz S, Reps T (1992) The Use of Program Dependence Graphs in Software Engineering. ICSE'92, Australia.
16. Binkley D, Horwitz S, Reps T (1995) Program integration for languages with procedure calls". ACM Trans. on Soft. Eng. and Meth. Volume 4 No 1 pp. 310-354.
17. Kruchten, P.: 'Architectural Blueprints - The 4 + 1 View Model of Software Architecture', *IEEE Software*, 1995, 12, (6), pp. 42-50.
18. Kruchten, P.: '*The Rational Unified Process: an Introduction*' (Swedish edition, Boston, Mass., London: Addison-Wesley, 2002).
19. Nicolas, J. and Toval, A.: 'On the Generation of Requirements Specifications From Software Engineering Models: A Systematic Literature', *Information and Software Technology*, 2009, 51, (9), pp. 1291-1307.
20. Jackson, M.: 'Automated Software Engineering: Supporting Understanding', *Automated Software Engineering*, 2008, 15, (3), pp. 275-281.
21. StarUML Project Team.: 'StarUML: User Guide' [online]. Available from: http://staruml.sourcefo rge.net/docs/user-guide(en)/toc.html [Accessed 15/1/2009].
22. ArgoUML Project Team: 'ArgoUML: User Manual' [online]. Available from: http://argouml-stats.tigris.org/documentation/manual-0.28/ [Accessed 15/1/2009].
23. NetBeans UML Plugin Team: 'NetBeans UML Plugin' [online]. Available from: http://netbeans.org/features/uml/index.html [Accessed 15/1/2009].