Utilising Refactoring To Restructure Use-Case Models

Ayman A. Issa

***Abstract*—Use case refactoring is one of the recent software engineering techniques that aimed at synthesising and refining use case models. Two new types of use case refactoring are proposed in this paper. First, behavioural refactorings aimed at synthesising the presentation and understanding of the described services. Second, structural refactorings aimed at refining and simplifying the different relationships between the following pairs: (1) concrete and abstract use cases, and (2) use cases and actors. The application of the proposed refactorings on a real use case model showed that the advantages of use case refactorings are not limited to the target use case models only, but on their relationsips with other software engineering artefacts. These include the facilitation of the extraction and utilisation of (1) use case patterns, (2) software metrics, and (3) software cost estimates from use case models. Further work is being carried out to automate the process of use case refactoring and integrate it with the underlying software development process.**

***Index Terms*—Use Case Model, Refactoring, Software Patterns, Software Metrics.**

I. INTRODUCTION

Use case modelling is an approach invented to capture requirements from the perspective of how the user will actually use the system instead of the perspective of the technical features that the system is required to incorporate [1]. This simple concept means that the requirements documentation can also be used as the basis for subsequent software development activities. Theoretically, this has been adopted by the unified software development process [2] which stated the use case model as a core model that drives the derivation of all subsequent software development models: design, process, implementation, and deployment models. Among other things, use cases construct the basis for [3]: defining functional requirements and objects, allocating functionalities to objects, defining object interaction and object interfaces, the user interface, test cases, determining development increments, and composing user documentation and manuals. Practically, this was hindered by a number of use case modelling problems [4]. Namely, use case granularity, inconsistency, and ambiguity [5]. Unfortunately, the current use case modelling methods give no guide to use case specifiers about how detailed and concrete the specifications of use cases needs to be, and how specific the scope of each use case should be.

Refactoring, which is a behaviour-preserving transformation [6], was originally introduced to structure and modularise source code. Subsequently, several authors have articulated and presented refactoring techniques in several fields in the software industry [6,7]. The most popular software refactoring fields include, but are not limited to, architecture and use case model refactoring.

The use case modelling literature considers two main use case model refactoring perspectives. First, Rui and Butler [6] addressed the modelling perspective of use case refactoring, whereas Metz et al. [7] discussed the relational perspective of it. Both use case refactoring paradigms resulted in an enhanced maintainable and understandable use case models. This raised a number of research questions to which this paper aimed to answer: (1) can use case refactoring participate in reducing the main use case modelling problems (e.g. granularity and consistency)?, (2) what aspects of software engineering can be supported by use case refactoring?, and (3) to what extent can these supported software engineering aspects facilitate the practical application of use case model based software development?

In section 2, the related use case refactoring literature is investigated. Section 3 presents the newly proposed behavioural use case refactorings. The relationship between use case refactoring and other software engineering paradigms is explained in section 4. In section 5, the proposed use case refactorings are demonstrated by example. Finally, the conclusion and future work are discussed in section 6.

II. USE CASE REFACTORING TECHNIQUES

A. Use Case Modelling Refactoring

Due to the diverse use case modelling semantics between practitioners, Rui and Butler [6] proposed a generalised use case metamodel based on following main use case dependency relationships: subset/combines, uses/includes, precedes/follows, requires, extends, is-specialization-of, resembles, and equate. Consequently, they utilised their proposed metamodel to adapt source code refactoring in use case modelling. The result was a set of use case refactorings that are classified into five main categories: creating a use case entity, deleting a use case entity, changing a use case

Table 1: Use case refactoring examples [6].

Refactoring Category	Example Refactoring
Creating a use case entity	create_empty_usecase create_empty_actor
Deleting a use case entity	delete_unreferenced_usecase delete_unreferenced_actor
Changing a use case entity	change_usecase_name change_actor_name
Moving an element of use case	move_actor_to_parent_usecase move_actor_to_child_usecase
Distributing behaviour	decompose_usecase decompose_goal

Ayman A. Issa, Software Engineering Department, Faculty of Information Technology, Philadelphia University, P.O. Box 1, Amman. 19392, Jordan, aissa@philadelphia.edu.jo

entity, moving an element of a use case, and distributing behaviour. Table 1 presents sample refactorings of each category.

B. Relational Use Case Refactoring

Metz et al. [7] discovered a gap between the definition of use case relationships in UML and the way practitioners refactor textual and graphical use case models using include and extend relationships. Examples of these differences include the usage of extend relationship to express both partial and fully parallel interaction courses and exceptional alternative use case behaviours. Therefore, Metz. et al. [7] proposed a detailed UML refactorings to fully support these practical needs. Their proposal was introduced as a set of changes to the UML: (1) abstract syntax behavioural package, (2) metamodel element use case, (3) metamodel element include, and (4) metamodel element extend. Examples of the alternative courses that are supported in Metz et al proposed refactorings are presented in figure 1.

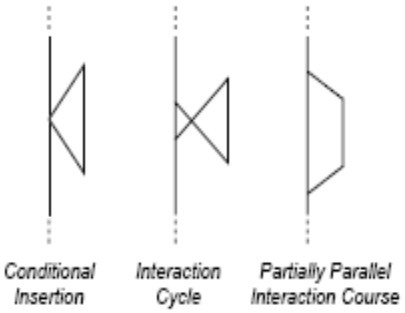

Figure 1: Metz et al. supported alternative courses [7].

This shows that Metz et al. relational refactoring is intended to extend UML capabilities. On the other hand, Rui and Butler use case refactorings are mainly intended to improve the presentation and readability of software systems use case models so as to facilitate further use case based software development. However, the analysis of use case modelling state of the art [5,8] suggests the need for other behavioural and structural use case refactorings so as to promote an integrated use case based software engineering. This includes both software development and management. Hence, this paper proposes a number of behavioural and structural use case refactorings that motivate better utilisation of use case models as detailed in the following sections.

III. THE PROPOSED BEHAVIOURAL AND STRUCTURAL REFACTORINGS

In building use case models, it is possible to apply two principally different approaches [9]: (1) top-down and (2) bottom-up. A top-down approach starts with identifying a number of use cases that are further refined with respect to their structural properties. A bottom-up approach starts with concrete examples of usage scenarios that are further generalized and synthesised into use cases that encompass these scenarios and more. Regardless how the use case was identified and modelled, the functionality of any single use case must be a complete course in itself. Similar use cases can begin in a similar way; it is not always possible to decide what use case has been instantiated until it is completed. Hence, similar starts can be factored out in a general reused use case with the differences in extending use cases.

Also, any use case scenario can be classified as [1,7]:

1) main scenario that describes the usual way in which the task is successfully performed,
2) variant scenario that describes another way of using the system where it is assumed that all steps execute successfully,
3) exceptional scenario that describes a scenario where exceptional or error conditions may arise,
4) recovery scenario that describes a scenario to recover from the exceptions and therefore successfully complete the task, or
5) failure scenario that describes a scenario where it may not be possible to recover from an exception.

Some use case specifiers prefer to separate exceptional, recovery, and failure scenarios in independent extending use case. Also, McCabe number, as cited in [8], can be calculated for each use case to indicate its complexity. Exceeding a specific threshold, 3 flows per use case as calculated by Issa et al. [10], representing the average number of paths (scenarios) per use case raises the flag for the possible need of some refactoring to simplify the use case.

In a proposed unified generic use case model, Issa et al. [11] showed that each use case scenario should be accompanied with a priority to facilitate using use cases in determining the contents of each development increment. Similarly, this priority property can be utilised as a refactoring indicator to separate highly important scenarios in related use cases when possible.

The existence of different use case specifiers in the development of one use case model have led to a different use case level of details as well as the use of different representation styles [5]. However, there should be a published style guide for in-house use cases. Letting every software engineer "do his/her own thing" increases the number of errors, makes testing much more difficult, limits the possibility of use case reuse, and decreases the overall efficiency of the project. Hence, the very last step of developing use case models should be devoted to the restructuring and unifying of the use case specifications to comply with the published in-house style guide.

Table 2: The newly identified use case behavioural and structural refactorings.

Refactoring Category	Refactoring Activity
Behavioural	reordering interaction steps reverse condition substitute Algorithm change unidirectional communication to bidirectional communication consolidate duplicate conditional fragment
Structural	merge (inline) two UCs change include relation to extend relation change extend relation to include relation extract general behaviour (include, extend, super, or sub UCs) collapse use case hierarchy collapse actor hierarchy expand actor hierarchy replace condition with extend relationship reformat to the in-house style guide

The congregation of all these traditional use case modelling activities guided the author to identify a number of implicit behavioural and structural refactorings as summarised in table 2. Compared to Rui and Butler presentational use case refactorings, the newly proposed use case refactorings should participate in creating a better structured and presented use case model. In addition, the application of the newly proposed use case refactorings may lead to uncover missed/lost details and other hidden use cases.

Although use case refactoring may lead to a better structured use case model, too many refactorings are difficult to control and actually diminish their specific effectiveness. Hence, refactoring misuse and overuse should be avoided to avoid their direct use cases inconsistency and explosion problems, respectively.

IV. IMPLICATIONS OF USE CASE REFACTORING IN SOFTWARE ENGINEERING

As use cases represent the main source of functionality in use case based software development, project managers believe that the use case model gives some indication of the final system size and the amount of effort involved in developing the corresponding operational software system. Therefore, a number of simple early applicable use case based software cost estimation models [11,12] have been developed with competitive accuracy results when compared to the more complex traditional software cost estimation models such as COCOMO II [12,13]. However, the accuracy of the results obtained from these simple models is highly affected by the diverse use case modelling misconceptions and problems discussed in previous section. For example, the number of actors (use cases) affects the estimates by combining actors (use cases) with similar descriptions into one actor (use case), the super actor (super use case). This increases the precision of the estimate and hence counting the actors (use cases) only once. Another factor that affects estimation is the level of detail that is involved in the use case specifications. The number of interactions within that use case measures the size of each use case, which consequently has a direct impact on the estimate obtained by the use case based estimation models. The suggested use case refactorings in the previous section are believed to participate in solving most of these use case modelling problems to result in a highly well-structured use case model. This will facilitate the extraction of a more accurate use case based metrics; and consequently, more accurate corresponding software cost estimation.

A use case pattern [10] is the design of a generic abstract use case representing a common solution to a common problem in a given context. A use case pattern for a specific class of applications consists of a general model and a collection of general steps such that for a given step the general model suggests one or more steps that can follow it [10]. Therefore, the skill most necessary for identifying reusable use case patterns is the ability to abstract. The proposed set of structural use case refactorings in the previous section (e.g. extract general behaviour, and replace condition with extend relationship) can be considered as a main source for identifying candidate generic use case patterns in their different three categories as classified by Issa et al. [10]: (1) project dependent, (2) application domain dependent, and (3) application domain independent. The identification of use case patterns in the early stages of software systems development showed many signs of success including: (1) the acceleration of the development process as it promotes a broad range of subsequent reuse for analysis and design patterns, (2) improved software engineers productivity, and, most importantly, (3) improved software quality. Further, Issa et al. [10] reported that use case patterns can be utilised in estimating software projects using their bottom-up use case patterns based estimation approach.

Finally, this shows the importance of use case refactoring as a pre-request to utilise the developed use case models in subsequent development phases including software measurement, software cost estimation, and building use case patterns catalogues.

V. USE CASE REFACTORING BY EXAMPLE

"Web Registration" project [14] has been utilised to explain the proposed use case based refactorings. The aim of this project is to develop an online student registration system. The initial use case model of the project consists of 11 use cases that detail the different features of the system. The "Submit On-line Registration Form" use case has been selected to demonstrate the newly proposed use case refactorings. Figure 2 presents the detailed specification of the selected use case.

The analysis of the "Web Registration" system use case model, in general, and the "Submit On-line Registration Form" use case, in particular, showed the necessity of a number of use case refactorings to synthesise and refine the overall structural and behavioural aspects of the use case model. Examples of the required refactorings include: (1) reordering interaction steps, (2) extract general behaviour, (3) replace condition with extend relationship, and (4) reformat to the in-house style guide.

Use cases represent the anticipated automated version of the underlying business processes. Some interactions ordering mismatches were found in the developed use cases. Hence, this necessitates the usage of the proposed "reordering interaction steps" refactoring to match the interactions order in both business and system processes. Also, the use case model development team consists of 4 use case specifiers which required the application of the "reformat to the in-house style guide" refactoring to eliminate styles and level of details differences. Figure 2 presents the re-ordered and re-formatted version of the on-line registration use case.

Use case precondition(s) describe(s) the system context in order to be able to start the use case. Hence, they may be represented as a general use case that aims to evaluate the system status before starting the current use case. This can be achieved using "extract general behaviour" structural refactoring. In the "Submit On-line Registration Form" use case, the checking of its five preconditions can be factored out in an independent use case that's included in the parent use case.

Similarly, alternative exceptional flows that are dedicated to check and validate user input are common in different use cases. Hence, they can be separated in an independent use case using the "replace condition with extend relationship" structural refactoring. Figure 3 presents the structurally refactored On-line registration use case.

- **Name:** Submit On-line Registration Form.
- **Goal In Context:** Teacher fills out the on-line registration form for students by entering each student's information. Information includes name, e-mail address, phone number and choosing classes of his/her interest from pull-down menus. The form is then submitted.
- **Primary Actors:** Teacher.
- **Priority:** High.
- **Preconditions:**
  a. User has collected classes of interest for world language day event from students.
  b. User has student information such as name, e-mail address, and phone number.
  c. User submits on-line registration form only during the registration period.
  d. School has paid registration fee for students.
  e. User logs on to the on-line registration page.
- **Main Flow:**

Step	Actor	Action Description
1.	System	Program displays the on-line registration page.
2.	Teacher	User adds, edits, or deletes students' information.
3.	System	Program checks for errors.
4.	System	Program displays updated student information on the bottom of the page with a small check box in front of it.
5.	Teacher	User repeats step 2-3 to add, edit, or delete other student's information.
6.	Teacher	User clicks on the submit button.
7.	System	Program displays a message saying that the information has been successfully received.
8.	Teacher	User clicks on the return button to go back to the on-line registration page or clicks on log off button to log off.

- **Alternative Flows:**

Action Description
2a. User enters invalid student information and clicks on the add or edit button.
2b. User does not enter some of the required student information fields and clicks on the add or edit button.
2c. User does not enter any student information and clicks on the add or edit button.
2d. User does not choose a class of interest for a student, from some or any of the pull-down menus, and clicks on the add button.
2e. User does not check any check boxes to edit student information and clicks on the edit button.
2f. User does not check any check boxes to delete student information and clicks on the delete button.
6a. User does not add any student information and clicks on the submit button.

- **Postconditions:**
  a. Student information is sent.
  b. Each student's information is displayed in the bottom of the on-line registration screen with a small check box to the left of it.
  c. Students' data are stored into the database.

Figure 2: On-line registration use case specification.

Finally, the extracted common general behaviours represent the main source for the identification of the different types of use case patterns. For example, precondition evaluation use case is a general idea that can be forecasted to all use case models as a domain independent use case pattern. However, the low level conditions and evaluations of each use case are dependent on the specific calling use case. Hence, a project dependent use case pattern of the evaluation use case can be created to represent the specific conditions. The same concept of the application domain independent and project dependent use case patterns

- **Name:** Submit On-line Registration Form.
- **Goal In Context:** Teacher fills out the on-line registration form for students by entering each student's information. Information includes name, e-mail address, phone number and choosing classes of his/her interest from pull-down menus. The form is then submitted.
- **Primary Actors:** Teacher.
- **Priority:** High.
- **Preconditions::** execute "evaluate system status" use case.
- **Main Flow:**

Step	Actor	Action Description
1.	System	Program displays the on-line registration page.
2.	Teacher	User adds, edits, or deletes students' information.
3.	System	Program checks for errors.
4.	System	Program displays updated student information on the bottom of the page with a small check box in front of it.
5.	Teacher	User repeats step 2-3 to add, edit, or delete other student's information.
6.	Teacher	User clicks on the submit button.
7.	System	Program displays a message saying that the information has been successfully received.
8.	Teacher	User clicks on the return button to go back to the on-line registration page or clicks on log off button to log off.

- **Alternative Flows:**

Action Description
2, 6: Execute validate user input use case.

- **Postconditions:**
  a. Student information is sent.
  b. Each student's information is displayed in the bottom of the on-line registration screen with a small check box to the left of it.
  c. Students' data are stored into the database.

Figure 3: Refactored on-line registration use case.

can be applied to the alternative exceptional flows general use case.

VI. CONCLUSIONS AND FUTURE WORK

A new set of behavioural and structural use case refactorings was introduced. The application of the proposed set of refactorings to use case models participated in synthesising and refining their structural and presentational aspects. Also, it has been shown that the application of the proposed use case refactorings paves the way to achieve the theoretical objectives of the unified software development process. This includes the utilisation of the use case model as a core model to drive subsequent phases of software development.

Furthermore, refactored use case models have shown the ability to support other software engineering activities such as software measurement, software cost estimation, and the identification of use case patterns. These findings participated in addressing the main research questions that was additionally supported by the application of the proposed refactoring approach on the selected case study: the "Web Registration" project.

Further work is being carried out to identify more use case refactorings that facilitates the development of highly structured use case models. A multi-steps identification approach is adopted to survey the use case modelling literature so as to identify any implicit refactoring activity. Also, source code development approaches [8] are being analogised to the use case modelling approaches to identify any similarity that may lead to adapt some source code refactorings to the use case modelling industry. This will be

followed by a detailed validation phase for the identified refactorings using a number of real use case models. Further dependency relationships are expected to emerge between other software engineering aspects and use case models.

Finally, a prototype tool is planned to be developed to facilitate the application of current and prospective use case refactorings. This should also accelerate the development of use case models and integrate them with the other software engineering artefacts of the underlying software development process.

REFERENCES

[1] Rumbaugh, J. , Jacobson, I. and Booch, G., *The Unified Modeling Language Reference Manual*. Reading, Mass.; Harlow: Addison-Wesley, 1999.

[2] Jacobson, I., Booch, G. and Rumbaugh, J., *The Unified Software Development Process*. Reading, Mass; Harlow, England: Addison Wesley Longman, 1999.

[3] Kirner, D., Porter, R., Punniamoorthy, P., Schuh, M., Shoup, D., Tindall, S. and Umphress, D., Extending Use Cases Throughout the Software Lifecycle *Software Engineering Notes*, 24 (3), 1999., pp.66-68.

[4] Lilly, S., Use Case Pitfalls: Top 10 Problems From Real Projects Using Use Cases *in Proceedings of Technology of Object-Oriented Languages and Systems - TOOLS 30, 1-5 Aug. 1999* Santa Barbara, CA, USA IEEE Comput. Soc,, 1999, p.174.

[5] Firesmith, D., Use Case Modeling Guidelines *in Proceedings of Technology of Object-Oriented Languages and Systems - TOOLS 30, 1-5 Aug. 1999* Santa Barbara, CA, USA IEEE Comput. Soc, , 1999, pp.184-193.

[6] Rui, K. and Butler, G., Refactoring Use Case Models: The Metamodel *in Proceedings of the twenty-sixth Australasian computer science conference on research and practice in information technology* Darlinghurst, Australia, Australia Australian Computer Society, Inc., 2003, pp.301-308.

[7] Metz, P., O´Brien, J., and Weber, W., Use Case Model Refactoring: Changes to UML's Use Case Relationship [online]. Darmstadt University of Applied Sciences: Dept. of Computer Science. Available from: http://www.fbi.fhdarmstadt.de/frames/organisation/personen/w.weber/public_html/Publ/use_case_model_refacturing-internal_report.pdf [Accessed 11/11/2003].

[8] Sommerville, I., *Software Engineering*. 6th ed. Harlow, England ; New York : Addison-Wesley, 2001.

[9] Molina, J., Ortin, M., Moros, B., Nicolas, J., and Toval, A., Towards Use Case and Conceptual Models Through Business Modeling *in Proceedings of International Conference on Conceptual Modeling / the Entity Relationship Approach*, 2000, pp.281-294.

[10] Issa, A., Odeh, M., and Coward, D., Using Use Case Patterns To Estimate Reusability in Software Systems *Information and Software Technology*, 48 (9), 2006b, pp. 836-845.

[11] Issa, A., Odeh, M., and Coward, D., Software Cost Estimation Using Use-Case Models: A Critical Evaluation *in Proceedings of the 2nd IEEE International Conference on Information and Communication Technology: From Theory To Applications* Damascus, 2006a, Syria pp.1-6.

[12] Jones, C., Software Cost Estimation in 2002 *The Journal of Defence Software Engineering*, 2002, pp. 4-8.

[13] Royce, W., *Software Project Management: a Unified Framework*. Reading, Mass.; Harlow: Addison-Wesley, 1998.

[14] Weeks, M., Jimenez, J., Mahe, S., and Watenabe, M., Online Student Registration Project Use Case Model [online]. USA.: Central Washington University. Available from: http://www.cwu.edu/~weeksm/UseCaseModels.doc [Accessed 10/1/2005], 2004.