ABSTRACT
Volatility and rapid changing requirements are critical factors particularly in the dynamic environment of highly evolvable software systems. In a situation that the requirement is changed, chances might be increased for all the related requirements and other affected software artefacts (i.e. designs, codes, tests etc) need to be identified and to propagate the changes as well. Therefore, requirement change propagation is considered as a key process not only during changes implementation to remain software consistency, but also to prevent any upcoming substantial failures. This paper introduces an approach to simplify the tedious requirement change tasks to facilitate both engineers and users during software development and evolution. The focus lies first on the externalisation of requirement specifications and its explicit representation of conceptual model for volatile requirements. Next, the focus will be placed on the systematic process model specified using SPEM that offers to simplify the requirement change process. The applicability of the proposed metamodel is demonstrated using the industrial strength IFAMMS application.

Keywords: Volatile Requirement, Requirement Change Propagation (ReChaP).

1 INTRODUCTION
Producing software systems that are able to adapt themselves to rapid environmental and requirement changes has become a topical issue in software engineering research [1, 2]. It has become increasingly essential to support software community throughout the evolution of large-scale projects and more complex software systems. For that reason, we realise that the best practice to better plan for software evolution is to control and support the process of changing requirement. This is because, a large portion of total software life cycle cost is devoted to introduction of new requirements, and removal or modification of the existing requirements [3]. Focus on requirement change propagation research continues to grow as software evolves. As many previous works have pointed out, this is due to the rapid changing and volatile requirements in the dynamic nature driven by the business environment in software life cycle [4-7].

Hassan and Holt [8, 9] define change propagation as the “changes required to other entities of the software system to ensure the consistency of assumptions in a software system after a particular entity is changed”. In other words, it refers to a process of actually carrying out a set of initial modifications to the software components, and to re-establish the system’s consistency, by making a set of estimated consequent changes [10].

Most of the prior works have projected more complex ways, resulting to complicated solutions and unanticipated risks. Formerly, the manual process of change propagation attempted to propagate the next changes to the second components based on proposed set of impacts during the impact analysis phase and based on consulted advice from the guru. In addition to that, maintainers would also proceed based on their own knowledge and prior experience to identify the next components to be changed. The manual steps in managing the change request such as adding new requirement, enhancing current software or fixing the bugs will result to an initialisation of the next changes to other related components that depend on each other in the same environment [9].

The main goal of this work is to propose an approach that supports the software community during propagating changes from high-level requirements to the design level during the development of evolvable software. The related high-level requirement artefact is requirement specifications while the design level artefacts are Class and State-Chart diagrams. Our aim is to equip software engineers with better specifications in regard to the characteristics of volatile requirements by producing metamodel and software process for...
the development and implementation of software that continuously evolves upon time.

This paper is organised as follows. Section 2 describes in brief on the context background of the proposed Requirement Change Propagation or in short the ReChaP approach. Section 2.1 presents the overall concepts of the proposed metamodel. In the following Section 2.2, discussion will be made in details on the proposed systematic SPEM software process. Subsequent to that, Section 3 presents on the discussion on the case study result of IFAMMS application the status for current work, followed by the related works that also addressing the requirement change issues in Section 4. Finally, Section 5 concludes the whole findings of this paper and presents the future plans for the next steps of our works.

2 THE ESSENTIAL OF PROPOSED ReChaP APPROACH

The proposed ReChaP approach aims at improving and simplifying the requirement change propagation process to support requirement management in software development and evolution as a whole. In particular, the main goals of this work are to develop a product metamodel (theoretical specification) and to develop a process model (implementation specification). In the next subsections, we will be describing on these twofold contributions that provide an approach to simplify the change propagation process during requirement change execution to support software evolution.

The main goal of this research is to propose an approach that supports the software community during propagating changes from high-level requirements to the design level during the development of evolvable software. With regards to our work, in a situation whereby the requirement is changed, chances might be increased for all the related requirements and other affected software artefacts (i.e. designs, codes, tests etc) need to be identified and propagated towards the changes as well. However in the context of this work, the proposed ReChaP approach will only be focusing on propagating changes for high level requirements. The automated change propagation process will only consider the design traceability that links forward from requirements to the design. This relation, which is also known as inter-dependency or vertical traceability refers to the relationship between software artefacts from the different phases [20].

2.1 The Proposed Metamodels

In this section, we introduce the product metamodel and the concepts behind it’s structure. Our work considers on requirement change activities that concerns on the volatility aspects and managerial issue in software evolution. As softwares evolve, the number of requirement changes can increase exponentially starting from the inception to the service phase for the end users. The requirement that is highly possible to be changed is known as volatile requirement. Therefore, it is essential to externalising the volatility concerns in governing decisions especially for future management of unanticipated requirement changes. Additionally, ones may minimize the consequence of changes onto software system by separating volatile requirements from more stable requirements. As for that, we explore a metamodeling approach to conceptualise the model with an explicit representation for the volatile aspect of the requirement changes. The key idea behind the metamodeling approach is to allow the user to define the well-formed structural semantics of the model abstraction. Figure 1 visualises the main classes and their relationships in the proposed metamodel for overall ReChaP approach between the software design elements, class and state-chart diagrams metamodels. In our case, class diagram metamodel exhibit the for static part while the state-chart diagram metamodel exhibit the dynamic part for linking the volatile requirement metamodel.

The first element in ReChaP approach metamodel is requirement classifications that are useful to categorise the type of requirements. In this work, the set of all volatile requirements is classified into two basic types namely functional and non-functional requirements (performance, specific quality and constraint) that originally from a concern-based taxonomy of requirement proposed by [21]. The taxonomy is chosen because it focuses more on the system requirement rather than project requirement and process requirement.

Requirements specification and definition templates are the next critical elements in the ReChaP metamodel because they help software engineers to express the accurate requirement statements according to the pre-defined templates. In the context of requirement specifications, the reusable templates are very much helpful to the software analyst to write the requirement using the consistent language by choosing a suitable predefined template and filling in the gaps (placeholders).
Apart from that, it also offers a simpler way in identifying the different types of requirements. Thus, it is a good method of standardising the language used for expressing the specific types of requirements [22]. Table 1 presents the discussed requirement types and their corresponding definition templates and its examples. As for the templates clause, it describes the language used to express the requirement, and the "<   >"cell is the placeholder, which is the gaps to be filled in by appropriate keywords or terms express the requirement, and the "<   >"cell is the placeholder, which is the gaps to be filled in by appropriate keywords or terms.

Lastly, the third element in the metamodel also introduces the taxonomy of requirement changes to facilitate the change request. This taxonomy highlights the characteristics of volatile requirements in terms of the problems, reasons and the sources of changes proposed by [23]. We adapt the taxonomy by classifying each change request by its origin(source), the reasons why it change, as well as two primary type for requirement change; simple (modification) and complex (addition,
deletion). These simple and complex types are explain in next 2.2 section for evolution phase.

2.2 The Proposed Process Model

The second pillar in this approach is the software process model that presents profound descriptions on a systematic process flow of propagating the requirement changes in the software design. The process models are adopted using a well-accepted Software Process Engineering Meta-Model (SPEM) v2.0 by OMG [24]. In addition, the core activities of the proposed requirement change propagation process are described in the analysis, design and evolution phases. The processes could provide guidelines especially for software engineers to assist them in simplifying the change propagation process for evolutionary software development. The usage of SPEM specifications is very useful in describing the software development process in a formal manner. In addition to that, the defined processes will capture the three main components namely activities, roles (actors) and work products (software artefacts) that are generally related in the development and evolution of a software system.

The proposed process models characterise responsible people such as system users, system designer and analyst. It illustrates their specific roles in terms of use cases as well as the detailed activities that the people performing. Also, the process models also highlight all the associated software artefacts that are being manipulated in particular during the implementation of requirement changes. Table 2 depicts the icon representation in SPEM to illustrate the elements involved in modelling the process.

Table 2. SPEM Notations

<table>
<thead>
<tr>
<th>Elements</th>
<th>Icon</th>
<th>Elements</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>🚀</td>
<td>Work Product</td>
<td>📊</td>
</tr>
<tr>
<td>Process Role</td>
<td>⚙️</td>
<td>Document</td>
<td>📁</td>
</tr>
<tr>
<td>Phase</td>
<td>📲</td>
<td>UML Model</td>
<td>🖥️</td>
</tr>
<tr>
<td>Object Flow</td>
<td>← - →</td>
<td>Control Flow</td>
<td>← - →</td>
</tr>
</tbody>
</table>

From the previous experiences using SPEM, we initially found that it helps to describe the iterative phases and to identify which step in the iteration is inconsistent and problematic. This is proven by the following examples of SPEM specifications used to describe in depth on a process of propagating the requirement changes into software designs. The examples presents some activities in analysis, design and evolution phases.

Phase: Analysis
Activity: Analyse requirements
Process Role: System Analyst
Activity Parameters: {kind: input}
Work Product: Problem statement {state: revised}
Steps
- Step: Identify requirement
- Step: Rewrite requirement according to requirement types

Phase: Design
Activity: Create metamodel using GME
Process Role: System Designer
Activity Parameters: {kind, input}
Work Product: Metamodel elements {state: revised}
Steps
- Step: Define metamodel elements
- Step: Design metamodel

Phase: Evolution
Activity: Implement requirement change
Process Role: Software engineer / maintainer
Activity Parameters: {kind, input}
Work Product: Requirement change plan {state: revised}
Activity Parameters: {kind, output}
Work Product: Design Model {state: changed}
Steps
- Step: Review the change plan
- Step: Perform the changes

The flow of activities for each phase are summarised as illustrated in Figure 3. The consequent paragraphs emphasise the activities in detail for each phase.

Analysis Phase
Subsequently, the collected requirements will be classified into functional and non-functional types (performance, specific quality, and constraints). Then, the collected requirement specifications will be refined based on the pre-defined template. The template is helpful in expressing the requirements in a standard manner as well as it offers simpler ways in identifying the different types of requirements. The second pillar of this phase is to identify the meta-model elements from the produced analysis model previously. In addition, the system analyst should study how to use the GME software that will be used for next design phase. Lastly, the whole packages in the system should be carried out as a last step in this phase.

Design Phase
The main aim of the design phase is to design and to create the metamodel that is considered as a central aspect in the proposed ReChaP approach. In addition to that, the purpose of this phase is to design the packages interfaces and transform the analysis model created in analysis.
phase to the design architectural models. It starts by designing the metamodel elements from the previous analysis metamodel stage. The metamodel is then developed using the GME software. Next, the EBNF rule constraints are embedded into the metamodel. The rule represents the syntax and semantic of metamodel specification in formal definition. Consequently, the requirement specification will be validated and revised. In the meantime, the entire related interface for system packages is sketched and designed. The design models such as class diagrams and state-chart diagram are transformed from the analysis models.

**Evolution Phase**

Requirement changes must be identified, planned and controlled in a well-manner. The changes took place continuously, throughout the life cycle of software from analysis towards maintenance and evolution stages. In this study, the authors have classified the requirement change request into simple and complex types. Normally, the simple requirement change requests are initiated by the system users. Therefore, they could directly analyse the requirements and perform the change request since this simple kind of request does not require additional or removal of requirement specifications or design elements. In short, we could generally say that simple request is a type of
Modification changes. However, it is a different situation for complex requirement change request. This complex change is regularly performed by software engineer or software maintainer. Performing addition of new requirements and deletion of existing requirement are considered as complex changes. Therefore, the task is assist by the automated tool in order to investigate and implement changes, as well as to validate the changed requirement specifications after the propagation process. This is due to the complicated ways to generate new requirements and to remove required requirements into/from the established interconnected set of requirements that have been defined for the software system.

3 CASE STUDY APPLICATION AND RESULT

Integrated Facility and Asset Maintenance Management System (IFAMMS) is selected as a case study to assess the feasibility and practical challenges of the proposed approach of this work. IFAMMS is a computerised application system used to record, track, maintain and dispose assets used by Maintenance Unit of Asset and Construction Office in University Technology of Malaysia (UTM). The current system is flexible in supporting various and rapidly changing of customer requirements over time during its servicing phase. As shown in Figure 4, there are 4 main packages in the overall IFAMMS system namely registration, asset tracking, asset complaint and maintenance.

As can be seen in Table 3, it briefly shows the examples of requirement specifications for IFAMMS application (based on requirement types) that have defined based on previous requirement templates definition in Table 1. Therefore, any change scenarios in the requirement palette placeholders “< >” will reflect and propagate to the class and state-chart diagrams elements. As for the initial result, by using the examples in the Table 3, it is found that the Requirement Template Definition element from the metamodel helps in managing traceability to propagate changes from requirement to the software design.
4 WORKS ADDRESSING REQUIREMENT CHANGE

The ReChaP approach is motivated by Business Rules-driven Object Oriented Design (BROOD) approach that considers business rules as a part of volatile requirement that exhibit the characteristic for evolvable software. BROOD focuses on propagating change from business rules to object-oriented design elements namely class and state-machine diagrams. It also equipped by traceability information using metamodel between business rule and software design elements. This approach offers easier software process as guidelines to perform change requests during requirement implementation and evolution phase [2-5].

TraceChange by [6, 7] is another technique that looks into modelling and process guidelines to address the issue of requirement change. It also provide fine-grained conceptual trace model and process to create relations, to investigate the impacts, and to check the consistency after change implementation. However, it only focuses on functional requirements of embedded control systems. In contrast of ReChaP work, we propose to address for both functional and non-functional types of requirement in various application domains as discussed in previous case study section. In addition to that, TraceChange process guidelines focus on three roles which involve in performing change impact activities namely requirement engineer, project planner and maintainer. However, this is contrast to claims by [8], which change request for volatile requirement may derived from the different types of system user.

Trace-based Impact Analysis Methodology (TIAM) approach utilizes the information on traceability and work product’s attributes to determine the impact metrics for requirement changes. The attributes are complexity, effort to produce the work products, the development phase for each produced work products, and degree of influence for each source of change to their target. TIAM also applies fuzzy logic theory to realize the traceability graph. It provides the prediction and prioritization for impact analysis. Requirement changes will be grouped based on similar predicted impact. The groups will be ranking from low to high level impacts [9, 10]. The limitation of this approach is its prediction and prioritization analysis are applicable deliberately for change planning rather than to guarantee the change is implemented comprehensively. In contrast to our work, ReChaP approach is not only suit to assist software engineer to plan for change, but also the ensure the consistent traceability among the dependent artefacts after the change has been implemented.

5 CONCLUSION AND FUTURE WORK

This paper reports on our ongoing work to develop ReChaP approach for requirement change propagation. The proposed approach is aim to simplify the rigorous requirement change tasks in order to facilitate software users and engineers during software development and evolution. Currently, significant parts of the approach, that are designing the product metamodel and process model are already done. However, its still need to be refined and realised into the multiple domain of case studies and validated through the synthetic experiments using human subjects. At the meantime, the software tool based on the proposed approach is under the development process. In the near future, we plan to conduct the synthetic controlled experimentation to validate the SPEM software process model and come out with the static analysis on improvement of the overall approach which consists of those two main elements; namely metamodel and software process.

6 ACKNOWLEDGEMENT

This research is partially funded by the ScienceFund Grant by Ministry of Science and Technology (MOSTI) under Vote No.79265. The authors would like to thank MOSTI and Universiti Teknologi Malaysia (UTM) for their financial support, as well as to individuals for their involvement and invaluable feedbacks during the time that this research was being conducted.

REFERENCES


