Analyzing the Impact of Requirement Changing on Software Design

Article · November 2015

CITATION
1

READS
10,980

2 authors:

Nedhal Al-Saiyd
Applied Science Private University
33 PUBLICATIONS **179** CITATIONS
SEE PROFILE

Esraa Zriqat
Applied Science Private University
6 PUBLICATIONS **66** CITATIONS
SEE PROFILE

Analyzing the Impact of Requirement Changing on Software Design

Dr. Nedhal A. Al-Saiyd[1] **Israa A. Zriqat**[2]
Computer Science Department
Faculty of Information Technology,
Applied Science University, Amman Jordan
[1]nedhal_alsaiyd@asu.edu.jo [2]I_zriqat@asu.edu.jo

Abstract

During product development and evolution, the set of elicited requirements may be changed; where modifications to existing requirements or additions of new requirements may affect existing requirements. Some requirements changes may impact the success of the product within established schedules. As software projects and systems increase in size and complexity, the need is raised to identify predict and control the potential effects of requirement volatility on the architecture software design. This paper describes the impact of changing the requirements on the architectural software design based on risks and the corresponding affected areas of the developed systems. It explores the impacts of new or changing system requirements on existing and future system goals and objectives, and identifies the factors that may influence the software architecture design. It is found that early defined and traced functional, data, quality attributes and other non-functional requirements are positively influence to software systems successes, as it is essential to evolve the requirements during all phases of the project development process. Refactoring is used to make the design more modular and structured.

Keywords: Requirements Changes, Requirement Evolution, Architecture Design, Design Modularity, Traceability, Refactoring

1. Introduction

Software requirement engineering (RE) is an important development phase to elicite the requirements from the customers, analyze them and represent them into analysis models that are considered as input to architecture design. Requirement is a significant factor for the SDLC, which identifies a capability, characteristic or quality of any project, defines what different stakeholders need, and presents how system will complete these needs [1]. Although there are many design texts that become more powerful in defining a project's requirements at the initial stages of the project, fewer researcher and practitioners focus on the important role that requirements play during the design process. In engineering design, requirements play essential role in determining a project's success. Therefore, many design texts emphasize requirement elicitation, definition tools and methods at the initial stages of the project, while mismanaging the requirements changes and evolution of project's requirements will cause project failure [2].

Concerning the requirements changing, each time the design phase has defect and is needed to fix the faults, the RE needs to be repeated. One of main sources of requirements changes can be the requirements-design gap. Hence, system architecture can be ambiguous, costly, and risky, even if the evolutionary information, process and the architecture are well defined and understood. Software modularity reduces the cost of development and maintenance, but it has some restrictions to deal with requirements evolution. To reduce the changes effects, a design strategy is needed to support analysis and rationale for design decisions [3]. The space transformation to define and interpret requirements changes depends on transformation type and scope as: adding functionality, rules and/or constraints to convert quality requirements to functionality on component and impose architectural pattern, or restructuring to apply design pattern on component and impose architectural style. The scope of

architecture transformations extends to components as well as the architecture as a whole. The evolution of product line assets often involves a combination of architecture transformations [4].

RE changes are unavoidable through software development or software maintenance. RE lacks of systematic clear guidelines of how to elicit functional, non-functional, and data requirements in specific domain from the stakeholders;

- Requirement engineers face difficulties in identifying, prioritizing, classifying requirements into clusters, specifying and representing requirements in consistent analysis models.
- Non-functional requirements are often described design constraints, quality goals, and operational constraints. They have dependencies on functional requirements, data requirements, data management, and processing and can be related to one another with complex interrelationships.
- Also, there are no structured methods for tracing requirements and there are no conceptual models to find the interdependency among requirements.

These issues may impact negatively the quality of software design. According to Lehman's laws, when a software is evolved its complexity is increased and its quality is decreased [5]. All the decisions during forward engineering and reengineering take into consideration the dependencies between the requirements engineering, software architectural design, and software quality. Unfortunately, existing approaches do not sufficiently handle and manage these relationships.

To compromise these difficulties and their associated problems, a management approach is needed depending on early specifying requirement changing drivers, identify changing requirements and risks, validate requirements using traceability-based approach, identifying design defects after the requirements definition and specification, evaluate design decisions and models, and enhance architecture design and consequently improve project success.

Section 2 presents the literature review. Section 3 explains the software requirements changing sources and causes Section 4 presents the proposed methodology used to resolve requirements changes risks to enhance architecture design that include explaining the Proposed Requirement Change Management Process Mode (RCMPM), requirement traceability, architectural refactoring and choosing development process model. Finally, section 5 presents the conclusion.

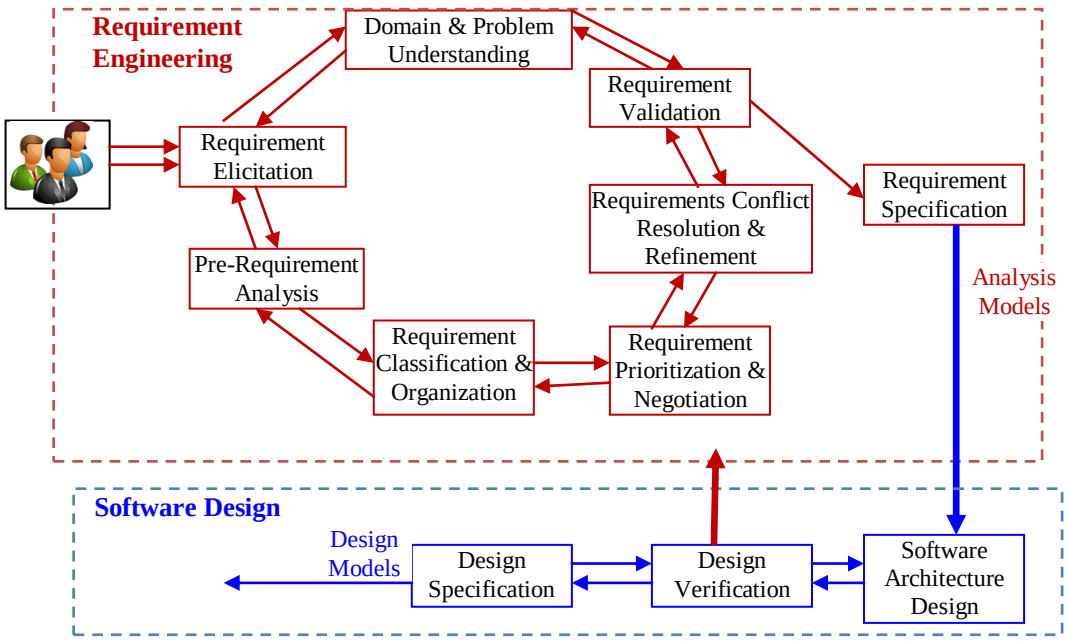

Figure 1 The requirement engineering and software design phases

2. Literature Review

Many studies have been performed on requirements evolution throughout a software project and introduced that over 80% of the defects occur in the verification of internal structure during software development. It is a challenging issue of requirements evolution and its impacts on problem solution, software design and the design quality [6], [7], [8]. Few researchers have been discussed the important role of the definitive requirements changes, on data design, architecture design, interface design and component-level design, while many change impact analysis techniques are done at the code level of software systems.

In 2002 Zhao Jianjun [9] studied the architectural-level impact analysis rather than the implementation-level impact analysis and presented an approach for changing impact analysis to support the software architecture in software evolution depends on architectural slicing method. The main idea of the approach is to assess the impact of changes on software architecture using formal modeling of software architectures specification. Therefore, the architecture description languages (ADLs) used to represent change impact analysis, to define high-level software structure as a collection of interacting components, and reason about system properties. The architectural structure is described as graph of computational components with set of interfaces called ports, configuration defined by instances and connectors among components.

In evolving large and long-lived systems, the requirements change is an important prerequisite for understanding the nature of requirements volatility.

During the development process, the rate of requirement changing was high at the completion time of requirements specification [10]. A qualitative method is developed to differentiate and evaluate requirements changes based on identifying the general types of changes, the root causes of changes and reasons for requirements changes. After collecting data and observations, the taxonomy is defined to classify changes, where the taxonomy consists of: Change Type, Reason, and Origin.. The method is applied to better understand the change process, analyze changes request average rates, and manage requirements changes in a large software development company.

In the maintenance of component-based software architecture, the component interaction-based approach is investigated to support dynamic change impact analysis. A set of impact rules are suggested to determine and perform changes in component and among components [11].

In 2008 Safoora Shakil Khan [12] showed that the increasing of requirements evolution is correlated to understand the connection between evolving conventional categories of requirements dependencies and their impact on the architectural decomposition. The evaluation results from an exploratory study showed the impact of typical changes in requirements dependencies on architecture design modules and interfaces. They discussed six types of dependencies; Goal Dependency, Service Dependency, Conditional Dependency, Temporal Dependency, Task Dependency, Infrastructure Dependency. There is no strict separation between the dependencies. They have analyzed how co-existing dependencies evolve, which dependencies have dominant impact during change and led to weak or strong interconnection of architectural elements.

In 2011 Matthias Riebisch [13] indicated that software architectures play an important role in the software development and software evolution processes. Software architecture transforms from requirements and project quality goals in problem space to technical implementation in solution space on the other side to enable long-term evolution of the software systems by expressing explicitly the design knowledge and fundamental design decisions. Such transformation defines dependency relationships between the requirements engineering, software architectural design, and software quality. The dependent features, architectural components, and interfaces have to be changed, if quality goals or requirements are changed. Software architectures help to manage the complexity of the software, enable evolution, and support changes in a well-organized way. The transformation is performed as a layered structure called Goal Solution Scheme (GSS), which supports forward and reverse engineering. This mapping considers the impact of architecture solution elements: design patterns, styles, frameworks, building blocks, and tools, on the quality goals. Reengineering and refactoring are connecting to the quality goals. The reengineering and refactoring tools have used to support the GSS scheme.

In 2011 Duong Thi Anh Hoang [14] explained a requirements-based methodology, which was developed to support the impact analysis of on-demand warehouses (DWHs). It formally described the

semantics of dependency between requirements engineering and architectural modeling, addressed the dynamic impact analysis and identified the traceability among different DWH layers. The methodology can overcome the problems concerning various business requirements by bridging the semantic gap between the requirements, the components and the architecture design of DWH layers. It helped to facilitate tracing of requirements, which have impacts on the architectural design and therefore to increase usability of DWHs by end users. They found that reusing DWH models play an important role in efficient DWHs development, management and in reducing the modeling time.

In 2012 Pimentel, João, et al [15], the architectural models that implicitly contain information of functional and non-functional requirements changes are used to resolve co-evolution problem. The co-evolution reasoning is performed using a single model to assess the impact of requirements and architecture and to respond to changes. The architectural description language (ADL) is considered as the set of reasoning mechanisms, which described the architectural components, connectors, interfaces, configuration, and internal elements. Reasoning supported forward and backward co-evolution for service-driven architectures.

In 2014 Joshua D.S. [2] analyzed weekly requirements documents of different team developers of different service-oriented design projects. The design requirements documents were updated weekly according to requirements changes of projects. The Function-Behavior-Structure model was used in for short-duration projects to explain the co-evolution of engineering design, and a set of guidelines and recommendations are developed.

3. Software Requirements Changes

3.1 Reasons of Software Requirements Changing

Requirements changes cannot be eliminated or avoided, and there are interdependent relationships between different software development phases. Software changes cause project time-consuming, cost, effort, quality and objectives more challenging issues. Therefore, it is important to take into consideration the interrelationships between the software requirements changes and the software design and other following development phases. Change-related aspects of software requirements have been studied. There are many reasons to change requirements during software project progress, where requirements changing are often inevitable during the software development lifecycle [16], [17], [18], [19]:

1. The requirements have ambiguity and the analyst misunderstood the customers input.
2. Some developers ignored some of the original requirements because they did not understand them or they are complex to implement them.
3. Some requirements are part of other defined requirements.
4. The same requirements are defined in different ways.
5. Customers change their minds and change their requirements.
6. New customers add new requirements through requirement elicitation phase.
7. The requirements need to be completed and involve all the proposed subsystems and adding new functionalities and features.
8. The scope of the software development projects is changed.
9. The errors in requirements specifications need be corrected.
10. The performance of the developed software need to be enhanced.
11. The proposed software product may need to be operated in different platform or new technologies.
12. Technical reasons.
13. Organizational restructuring and changing policies.
14. Mistakes made in the requirement elicitation phase are particularly costly and increase the product's defect density. cause
15. Poor communication and relationships between developers and clients and users
16. Inconvenient use of Analysis techniques.
17. Incompatible and low Requirement specification quality.
18. Lack or limited development team skills and experiences.
19. Lack understanding of project problem.
20. Size and complexity of the project under development.

3.2 Difficulties and Risks of Changing Requirements

The risk factors of requirements changes contain four essential factors concerning requirements change, namely: People, Process, Existing Software Product and Organization based on frequency analysis [9]. Every factor has its correlated risk elements that need to be considered and studied through the impact analysis for implementing requirements change. They are formulated in a hierarchical structure that shows the most important risk factors in the first layer and the risk elements in the successor layers which may cause the project failure, product low-quality, late delivery, and cost overrun [20]. They are adequately addressed in Figure 2, which represents a hierarchal structure for risk factors and their changes elements.

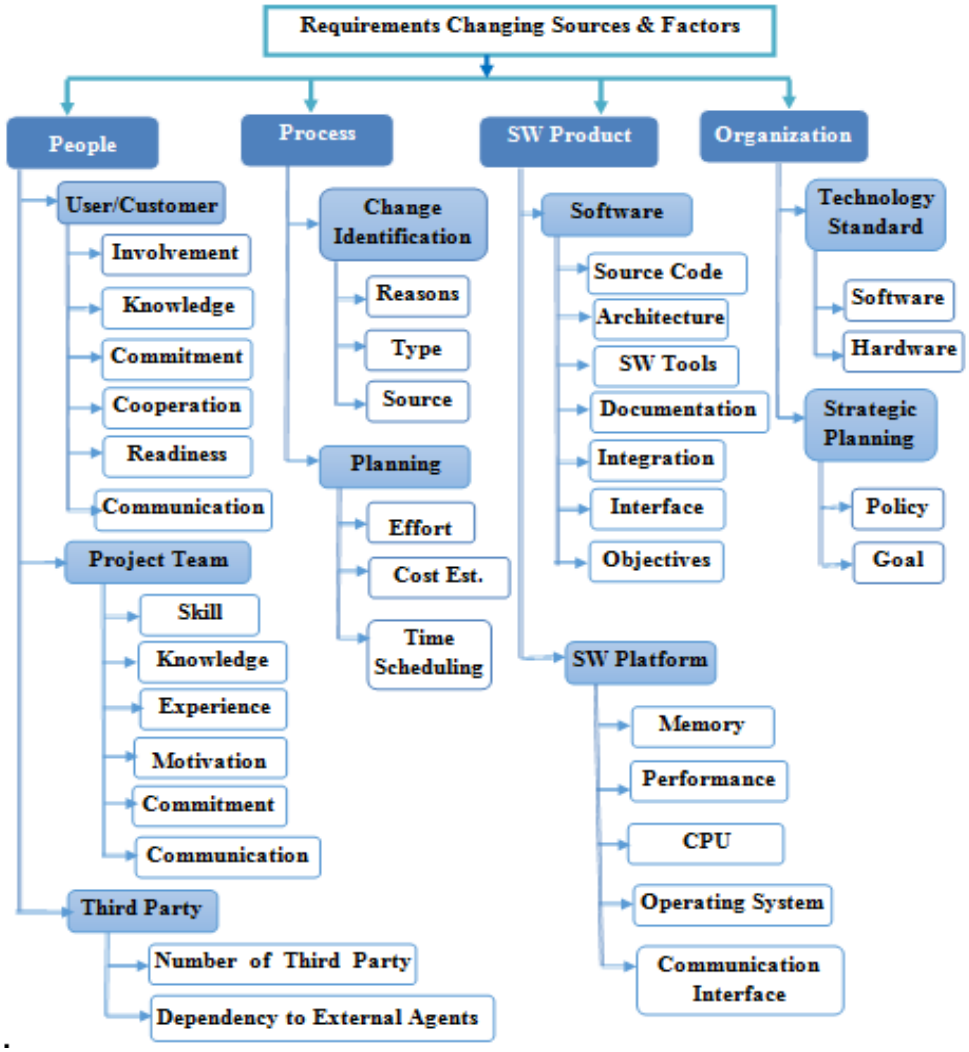

Figure 2 Classification of software requirements risk factors and their changes elements

4. Proposed Methodology

4.1 Handling Changing Requirements to Support Architecture Design

Assessing risks helps to show the interrelationships between requirement changes and their potential effects of unwanted conditions on the existing software, analyze them, and prioritize the risks. We need a set and actions and a mechanism used to successfully manage requirements changes in terms of time and cost, minimize risks, enhance the quality of software design, system value and user satisfaction. It can be done through:

a. Propose Requirement Change Management Process Model (RCMPM) to get precise, complete, consistent requirements specifications.
b. Reduce the considerable time for making rational design decision to choose the best of alternative designs to gain high quality as a consequence of that decision.
c. Control and adopt best design model.

d. Better stakeholder involving in requirement engineering and design phases, and
e. Better deal with requirements change and improve characteristics of software companies.
f. Select the appropriate incremental process model.

4.2 Proposed Requirement Change Management Process Model (RCMPM)

Early defined and traced functional, data, quality attributes and other non-functional requirements are positively influence to software design and consequently to software systems successes, as it is essential to evolve the requirements during all phases of the project development process. Well-identifying the non-functional constraints after requirement changing can help to identify the design patterns and the design architectural decisions that implement the volatile requirements, and increase design quality.

Since the mismanaging of project requirements leads to one of major causes of project failure, a Requirement Change Management Process Model is proposed. RCMPM tasks include:

i. Identifying the related documents and the requirements that are influenced by requirement changes.
ii. Identifying the corresponding customers to elicite the requirements.
iii. Identifying the types of requirements changes that may implement in the software.
iv. Reasonably analyzing the changes and their influences to decide the feasibility and efficiency of implementing them or not. The RCMPM tasks are not easy to perform.
v. Acquiring knowledge on how to perform a change.
vi. Verify the implementation and evaluate change's effect.

Figure 3 shows the UML activity diagram for RCMPM activities.

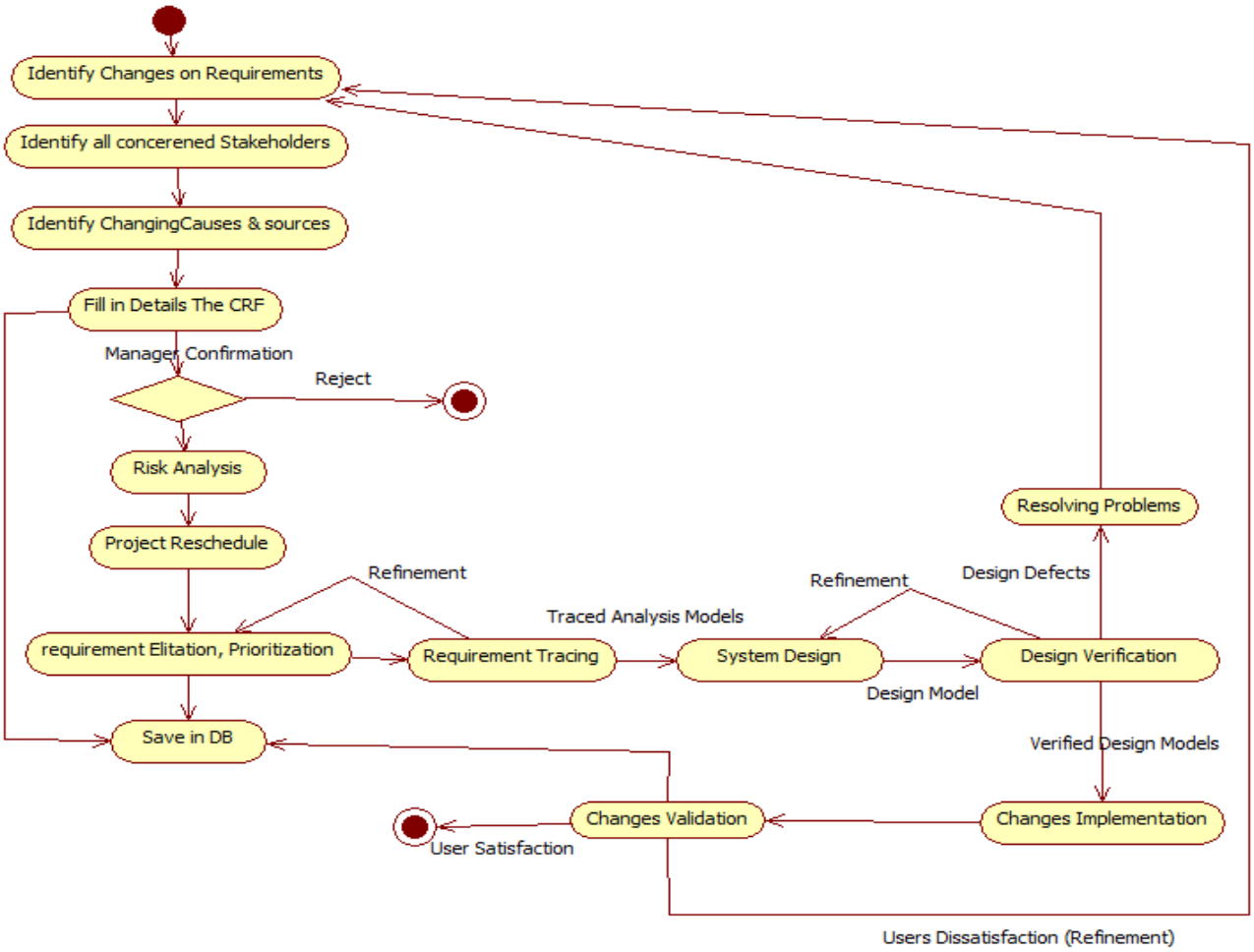

Figure 3 Activity Diagram of Proposed Conceptual Requirement Change Management

4.3 Requirements Traceability

The requirements traceability is an important technique in requirements management, which helps to validate the requirements to deal with changes at early stages of SDLC, to check the requirements for completeness and consistency. Requirement changes must be identified to ensure that all the stockholders requirements have been involved and satisfied [21]. Trace-based approach focuses on analyzing the following issues [22], [23], [24]:

a. Identify the conflicting requirements,
b. Check on requirements inconsistency,
c. Check on the changing incompleteness and imprecision of requirement specification at method, class, and package level,
d. Check on changing incorrectness,
e. Find the origin of each requirement change, trace back to the stockholders in requirements elicitation activity to prioritize the entire requirement,
f. Track the change that is done, and decide quickly the numbers of artifact that will be affected by a suggested changes,
g. Assess and specify risks on software design during development of project, which are influenced by changes, and
h. Estimate the cost, the amount of time, and effort spent on tracing each requirement depending on the priority.

All the above issues need to be well-studied and analyzed to produce a qualified requirements specification as a result of requirements tracing, to support better design.

Traceability matrix is represented as a table to show the logical relationships between individual functional requirements and other artifacts. Many software tools are available and used to trace requirements specification throughout the system development lifecycle; like the RequisitePro [23], which helps software developers to manage their requirements, to write good use cases, to improve traceability. Retro is another tool that has ability to assess an existing requirements traceability matrix to enhance the functionality and filter the display of candidate links among requirements. A tool by Rational, which have the ability to create a customized requirement types, and trace requirements to each other by type, or in hierarchical sequence and help to show the traceability from the highest-level requirement, down to the lowest level test cases.

Requirements traceability technique consists of two major activities: Pre-requirements specification traceability, which is concerned to the aspects of a requirement's life prior to its implication of the requirement specification, and Post-requirements specification traceability, which is interested to the parts of a requirement's life that result from requirement deployment and implication of the requirement specification. Figure 4 shows the main activities of requirements traceability after changing requirements to analyze the results of making change. Complexity metrics are taken in consideration to ensure not increasing the complexity metrics of software design to minimize the impact of changes and to achieve the goals of software product design. Requirement correctness, completeness, consistency and non-Ambiguity must achieve high-rating level that are not lowered than 75% for measuring the use case diagram and use case specification quality. In object-oriented approach, the traceability associates the problem domain knowledge; represented by of requirements specification, use case specification, use case diagrams, domain class diagrams and object models, to the software design knowledge; represented by component diagram, design class diagram, design object model, sequence diagram.

The overall tracing effort increases according to the increasing number of changed requirements, changed documentations and models, the tracing tools used to support the analyst work, complexity and size of change impact analysis.

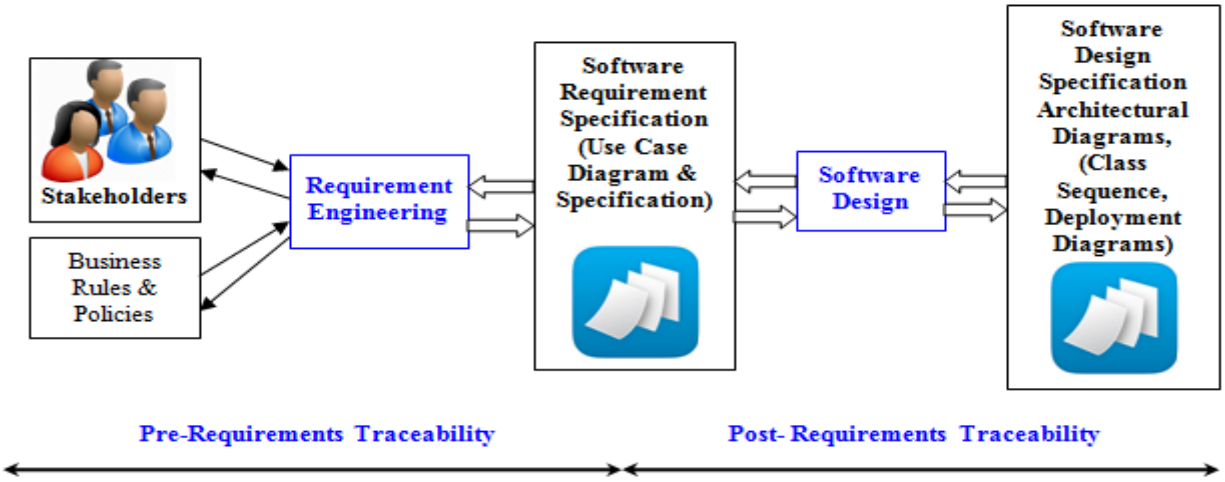

Figure 4: Software Requirements Traceability

4.4 Architectural Refactoring

Software architectures describe how a system is decomposed into components, how these components are interconnected and communicate with each other [27]. When software requirements are modified, they may negatively influence software architecture that is no longer adequate for the current requirements [28]. Changing software architecture requires changing the internal design structure [29]. Architectural refactoring is a controlled technique to modify the internal structure of software without changing its functionality and external behavioral properties. The time and frequency of refactoring depend on software developers. It is used to reduce the design structure complexity and enhance the maintainability of the software [25], [26], [30]. Refactoring is a human-driven technique and there are semi and fully-automate software refactoring tools that can support software designers in their work even they may take a significant time. Most Integrated Development Environments (IDE) provides tools to support design restructuring process without changing its functionalities, such as Eclipse, Microsoft Visual Studio, IntelliJ, Idea, Borland JBuilder, RefactorIT and jCOSMO [32], [33].

The traced requirement specification at the early stage of software development and the old design is needed as input for restructuring design, which is also called architectural refactoring. The requirements changes make restructuring more difficult when they influence the connectivity of multiple components, and interactions among components. Therefore, they affect the level of architecture metrics; concerning the structure complexity, modularity, coupling and cohesion. The interaction between software components, classes and among class methods characterizes the complexity of the design. The designer needs to change the internal structure to produce better design that has low-coupling and high-cohesion criteria to lower the structure complexity and make the software more maintainable. Refactoring is done using series of standard micro-refactoring tasks that do small changes on design and code to reduce the risks, improve the design, make the software easy to understand and maintain. Architecture styles and design patterns play an effective role in all refactoring tasks to analyze the changes on design components and to improve quality.

4.5 Development Process Model

If some of the changed requirements are highly ambiguous and not well-understood, the project is need to be executed into a series of small increments, and this leads to adopt incremental or iterative development process model as a strategy to cope this situation. Agile development process is preferred to use to get user feedback to confirm the system validity. The software domain, scope, size, type, application value, customer/user involvement and determine the appropriateness of development process.

5. Conclusion

- Software requirements engineering is a significant development phase that requires more and more expertise and knowledge from software developers and stakeholders.

- Early defined and traced functional, data, quality attributes and other non-functional requirements are positively influence to software systems successes, as it is essential to evolve the requirements during all phases of the project development process
- The stakeholders need to be involved with software requirement engineers in identifying the requirements defects, changing, adding new requirements, and requirement prioritizing.
- The stakeholders need to be involved and cooperated with software designers in making software architectural decisions of the loosely-coupled system components and providing the information to reduce the amount of work and cost.
- Well-identifying the non-functional constraints after requirement changing can help to identify the design patterns and the design architectural decisions that implement the volatile requirements, and increase design quality.
- Refactoring is used to make the design more modular and structured. The internal structure of software is improved after changing requirements improve the extensibility, modularity, reusability, complexity, maintainability of the new version of the system.

Acknowledgment
The authors are grateful to the Applied Science University in Amman, Jordan for the full financial support granted to cover the publication fee of this research article

References

[1] Sajjad, U., and Hanif, M.Q. 2010. Issues and Challenges of Requirement Elicitation in Large Web Projects, School of computing, Blekinge Institute of Technology Ronneby Sweden.

[2] Summers, J.D., Joshi, S., and Morkos, B. 2014. Requirements Evolution: Relating Functional and Non-Functional Requirement Change on Student Project Success. *In Proceedings of the ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, 17-20 August. Buffalo, New York USA.

[3] Felici, M. 2004. Observational models of requirements evolution (Doctoral dissertation, University of Edinburgh), School of Informatics

[4] Bosch, J. 2000. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach.Pearson Education.

[5] Lehman M. M. and Ramil J. F.. 2001. Rules and Tools for Software Evolution Planning and Management, *Annals of Software Engineering*, Vol. 11, No. 1, pp. 15-44.

[6] Chen, Z.Y., Yao, S., Lin, J.Q., and Zeng, Y., Eberlein, A. 2007. Formalization of Product Requirements: From Natural Language Descriptions to Formal Specifications. *International Journal of Manufacturing Research 2(3), pp. 362–387*.

[7] Pahl, G., Beitz, W., Wallace, K., and Blessing, L. 2007. Engineering Design: A Systematic Approach, Springer- Verlag London Limited, London.

[8] Joshi, S., Morkos, B., Shankar, P., and Summers, J.D. 2012. Requirements In Engineering Design: What Are We Teaching?," Tools And Methods For Competitive Engineering, I. Horvath, ed., Karlsruhe, Germany, p. no. 38.

[9] Zhao, J., Yang, H., Xiang, L. and Xu, B. 2002. Change Impact Analysis to Support Architectural Evolution. *Journal of software maintenance and evolution: research and practice*, 14(5): 317-333.

[10] Nurmuliani N, Zowghi D, Powell S (2004). Analysis of Requirements Volatility during Software Development Life Cycle, In: Proceedings Australian software engineering conference, pp 28–37. IEEE.

[11] Feng, T. and Maletic, I. J. (2006). Applying Dynamic Change Impact Analysis in Component-based Architecture Design. 7th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06)

[12] Shakil, K.S., Greenwood, P., Garcia, A., and Rashid, A. 2008. On The Impact of Evolving Requirements-Architecture: An Exploratory Study. 243-257. Springer Berlin Heidelberg.

[13] Riebisch, M. 2011. Problem-Solution Mapping for Evolution Support of Software Architectural Design. *In Proceedings of Software Engineering 2011*.

[14] Hoang, D. T. A. 2011. Impact Analysis for On-Demand Data Warehousing Evolution. In ADBIS (2): 280-285.

[15] João, P., Castro, J., Santos, E. and Finkelstein, A. 2012, January. Towards Requirements and Architecture Co-Evolution. *In Advanced Information Systems Engineering Workshops*: 159-170. Springer Berlin Heidelberg.

[16] Sharon, M., and Greer, D. 2009. Software Requirements Change Source Taxonomy. *In Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference:* 51-58. IEEE.

[17] Talha, J. and Durrani, Q.S. 2004. A study to investigate the impact of requirements instability on software defects. *ACM SIGSOFT Software Engineering Notes*, 29(3):1-7.

[18] Lars, M., Saarinen, T., Tuunanen, T. and Rossi, M. 2004. Managing Requirements Engineering Risks: Analysis and Synthesis of The Literature. *Helsinki School of Economics Working Papers W-379, 63.*

[19] Alejandro, L., Nicolas, J. Toval, A. 2009. Risks and Safeguards for The Requirements Engineering Process in Global Software Development. *Fourth IEEE International Conference on Global Software Engineering:* 394-399.

[20] Axel, V.L. 2000. Requirements Engineering in The Year 00: A Research Perspective. In ICSE '00. *In Proceedings of the 22nd international conference on software engineering*: 5–19. New York, NY, USA.ACM Press.

[21] Bashir, S.S. and Ahmed, S. 2008. Requirements Validation Techniques Practiced In Industry: Studies Of Six Companies. *Blekinge Institute of Technology, Sweden*.

[22] Muhammad, S., Ibrahim, S., and Mahrin, M. N. R. 2011. An Evaluation of Requirements Management and Traceability Tools. *World Academy of Science, Engineering and Technology, WASET.*

[23] Andrew, K. and Saiedian, H. 2009. Why Software Requirements Traceability Remains A Challenge. *CrossTalk The Journal of Defense Software Engineering* 22(5):14-19.

[24] Glenn A. S. 2001. Requirements Traceability and The Effect on The SDLC. Spring Cluster.

[25] Abebe, M., and Yoo, C.J. 2014. Trends, Opportunities and Challenges of Software Refactoring: A Systematic Literature Review. *International Journal of Software Engineering and Its Applications* 8(6): 299-318.

[26] Maddeh, M., Romdhani, M. and Ghédira, K. 2009. Classification of Model Refactoring Approaches. *Journal of Object Technology* 8(6):143-158.

[27] Soni, D., Nerd, R.L., and Hofmeister.C. 1995. Software Architecture in Industrial Applications. *17th International Conference on* Software Engineering - ICSE 1995, 23-30 April. Seattle, Washington, USA.

[28] Taiga, N. and Basili, V.R. 2005. Metrics of Software Architecture Changes Based on Structural Distance. *In Software Metrics, 2005. 11th IEEE International Symposium*: 24-24. IEEE.

[29] Vipin, S. and Kumar, S. 2012. Impact of Coupling and Cohesion in Object-Oriented Technology,. *Journal of Software Engineering and Applications* 5(09), 671.

[30] Guo, L.H.X. and Shao, W. 2013. Monitor-Based Instant Software Refactoring. *Software Engineering, IEEE Transactions on* 39(8):1112-1126.

[31] Baig, I. 2004. Measuring Cohesion and Coupling of Object-Oriented Systems - Derivation and Mutual Study of Cohesion and Coupling.

[32] Erica, M. and Strooper, P. 2006. Evaluating Software Refactoring Tool Support. *Proceedings of 17th Australian Software Engineering Conference*. 18-21 April 2006, Sydney, Australia, pp. 331-340. IEEE.

[33] Katić M., and Fertalj K. 2009. Towards Appropriate Software Refactoring Tool Support. *Proceedings of the 9th WSEAS International Conference on Applied Computer Science*, 140-145.

AUTHORS PROFILE

Dr. Nedhal A. Al-Saiyd. She got her B.Sc. degree in Computer Science from University of Mosul-Iraq in 1981, M.Sc. and PhD degrees from University of Technology, Baghdad-Iraq in 1989 and 2000 respectively. She is an Associate Prof. at Computer Science Dept., Faculty of Information Technology, in Applied Science University, Amman, Jordan. She has got more than 24 years of teaching experience. Her research interests include: Software Engineering, Ontology Engineering, Intelligent Systems, User Authentication, Security, Image Processing and Speech Processing.

Israa Zriqat. She obtained her B.Sc. degree in computer science from Yarmouk University, Irbid-Jordan in 2009, and now she is M.Sc. student in Computer Science from Faculty of Information Technology, in Applied Science University, Amman, Jordan. Her research interests include: Software Engineering, Artificial Intelligence, and Data Bases.