Computer and Information Science; Vol. 9, No. 1; 2016
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

A Multistep Approach for Managing the Risks of Software Requirements Volatility

Nedhal A. Al-Saiyd[1]

[1] Applied Science Private University, Jordan

Correspondence: Dr. Nedhal A. Al-Saiyd, Faculty of Information Technology, Applied Science Private University, Amman, Jordan. E-mail: nedhal_alsaiyd@asu.edu.jo

Received: December 6, 2016 Accepted: January 7, 2016 Online Published: January 31, 2016
doi:10.5539/cis.v9n1p101 URL: http://dx.doi.org/10.5539/cis.v9n1p101

The research is financed by (Applied Science Private University).

Abstract

Software is changed continuously in order to respond to different users and business needs. Requirements are changed dynamically to improve software usability and increase its value, but requirement volatility sometimes cause failures for many projects because of inadequate understanding of the changing causes and the consequences of these changes. This research highlights the importance of managing requirement changes, classify them, and control the impact risks of requirement volatility on software project. The proposed model is designed based on software requirements risks factors and how to reduce their impacts. Generally, requirements changing is considered as a difficult, costly and time-consumed task, and consequently it is too important to study the inter-relationships between the changes and their impacts on the other phases of software system. The good understanding of the changing causes and their consequences can improve and support requirements management process and also lead successfully to the predicted goals of changes. The high quality of the requirements influences the success of a software project during software development and maintenance processes.

Keywords: change control, impact analysis, requirements changes types, requirement management, software maintenance, volatile requirements

1. Introduction

Software is dynamic and is evolved continuously over time in order to respond to different customers, end users and business needs. Adding new requirements, deleting or modifying existing requirements may be required through software development process phases or at software maintenance, where demands for changes cannot be avoided (McGee & Greer, 2012). Requirements that are changed during the software development life cycle may occur at the design phase, coding, testing until the software product is produced (Nurmuliani, Zowghi & Powell, 2004). One fourth of requirements changes occur during maintenance phase, which is more costly than changes occur during development phase; because the software adapted to the new environment, business and end users requests (McGee & Greer, 2012), (Sudhakar, 2005). These changes have an influence on the design, code, test, deployment and their related artifacts that may require further work on them to keep the software work effectively and efficiently. Approximately 70 percent of software projects have failed to deliver the qualified and business-valued services, and caused user's dissatisfactions. Consequently, the cost estimation of requirement changes will be increased significantly at the latter phases of the software development (NIST, 2002).

Changing requirements may exist at the requirement engineering, analysis modeling, system design, design modeling, implementation phase, or at maintenance phase. Requirements volatility is a measure of how much requirements change through the software development life cycles and operational phase after software delivery. The traditional measure of requirement volatility is done by counting the modification, deletion, and addition of requirements, through certain period of time. Requirements volatility has relatively great impact on software project cost, schedule, resources and the quality of final product release (Sudhakar, 2005), (Thakurta & Ahlemann, 2010), (Dev & Awasthi, 2012). The software requirements changes can be classified as (Singh & Vyas, 2012):

1. **Pre-Release Requirements Changes:** it is referred to the requirement changes at the software development process. It can be implicitly classified into two sub-classes:
   i. **Pre-Functional Specification:** it involves the changing requirements during requirement elicitation, analysis activities of requirement engineering and before completing the functional specification.
   ii. **Post-Functional Specification:** it involves changing requirements during software design, coding, and testing, after completing the functional specification, and they are done by the customers and software developers.
2. **Post-Release Requirements Changes:** The changes occur in the maintenance phase, after the software product is deployed. It includes adaptive, preventive, corrective and perfective maintenance types.

Requirement volatility can be risky at software development and costly particularly at the late stages, since it may require re-engineering. The problem is with insufficient changing management data and systematic approach; not with requirements volatility. Identifying the problems help to minimize the risks, cost and poor communication between stakeholders and developer. In software development life cycle, a lot of work was spend on requirement changing, and changing management but not that much work has been done on minimizing the impact of requirement volatility (NIST, 2002), (Thakurta, 2010).

The misunderstanding of problem domain, lack of well-specified requirements, missing requirements, unavailability of practical structured methods to trace requirements, lack of requirements traceability templates, unavailability of conceptual models to find the interrelationships and dependencies among requirements, may negatively impact the quality of software design and consequently the quality of software systems. Most software system stakeholders found it almost impossible to understand how changes to a requirement will impact the overall system, since tracking the requirements changes need understanding their textual-based artifacts that consists of detailed requirements documents that are difficult to understand.

To compromise the difficulties and related problems, a comprehensive methodology and a management approach is needed to early identify changing requirements risks, follow changing on requirements particularly in the late stages of software development process, check requirements completeness and consistency, and enhance software quality processes to make them more responsive and more efficient. We need first to understand the factors that make changing difficult and costly.

The paper structure is organized as follows: Section 2 provides the related work on the subject matter. Section 3 briefly explains the reasons and causes of change requirements**.** Section 4 presents and elaborates the proposed comprehensive model and methodology. Discussion and key findings are discussed in section 5. Finally, section 6 concludes the paper with a summary that outlines the main findings.

2. Related Work

Research by (Bhatti, Hayat, Ehsan, Ahmed, Ishaque, & Z.Sarwar, 2010) statistically analyzed the changing requirements in each development phases. They found that the effects of changed requirements in earlier phases have significant potential impact on the following phases. The more changing requirements in RE, the fewer amount of changes are requested in design phase. The total effort of changing requirements at post-release phases of SDLC (i.e. maintenance) is higher than effort of changing requirements at pre-release of SDLC phases; as in designing, coding and testing. Requirements volatility usually occurred after proving them by customers at RE and it affect the estimations of cost, time, effort, the product quality and the success of the resulting product. It may cause negative effects on the later phases of software development (Thakurta & Ahlemann, 2010).

A careful choosing of software development model could lead to some control over the predicted volatility risks, where the characteristic of software process models influence the complexity, requirement management and the ultimate software objectives (Thakurta & Ahlemann, 2010).

(Nurmuliani and Zowghi, (Nurmuliani, Zowghi & Fowell, 2004) indicated that the major reasons of requirement volatility were the customer needs. Developers increasingly understand the software products, market demand and changes in the organization politics. There is a multifaceted relationship between requirement volatility and software project performance, which can be measured by developing and delivering the software product in time and within estimated budget assigned to the project.

However, the findings of (Zhao, Yang, Xiang & Xu 2002) in their paper confirmed the usefulness of the changing impact analysis in software maintenance and evolution, regression testing, and debugging. They studied the impact analysis of changes on the system design level; as a collection of interacting components, rather than the implementation details. To evaluate the effect of the changes on software design, they suggested architectural slicing technique to analyze the formal architectural specification. The analysis consists of three basic design entity types: components, connectors and the configuration topology. The cost-precision tradeoffs for program behavior are examined based on "Coverage Impact" and "Path Impact" dynamic impact analysis techniques. They compared several source codes concerning their execution time, space and costs, and found that "Path Impact" is more precise and more expensive than "Coverage Impact" (Orso, Apiwattanapong, Law, Rothermel & Harrold, 2004).

To control the change process, (O'Neal, 2001), (O'Neal, 2003) presented a traceability-based impact analysis methodology that determines the impact of changed requirements on SDLC. The changing requirements are classified according to their similarities and impact on software development work. The empirical studies showed that improving the requirement changing management increased the software productivity of the developers and improved the project planning and cost estimation (Damian, Chisan, & Thamsamy, 2005).

Researchers proposed a mathematical model to identify the requirements volatility; they found that the maximum changing rates depend on source code size and duration of requirements volatility, where the volatility is calculated from the function points and lines of source code in the software product (Kulk & Verhoef, 2008). A linear regression analysis models are proposed to help project managers in predicting the volatility of requirements of a medium-size software project and to minimize the risks caused by volatile requirements, like schedule and cost overruns. The correlation-based estimation model is presented to derive the size of changed requirements. Four dependent variables, number of actors, use cases, words, and lines; are extracted from requirements documents and applied into the prediction system. It can be used for all use case documents that are written in textual form (Loconsole, 2008). (Ferreira, Collofello, Shunk & Mackulak, 2009) showed that dynamic simulation model assist in well-understanding of the cost, schedule, and quality impacts of requirements volatility on the development of software project. The model is based on 50 parameters and used as an effective tool.

3. Reasons and Causes of Change Requirements

Software requirements are dynamically evolved and changed due to numerous and diverse reasons (McGee & Greer, 2012), (Sudhakar, 2005), (Dev & Awasthi, 2012), (Tripathy & Naik, 2015). They require further study to extract and propose others to enhance the new software release and the management decision making. In this work, we gathered, and studied 32 critical factors, and classified them into four main categories. They represent the potential risk areas that may lead to software success or failure and to estimate how much it might cost to implement the change. Also, they help us to explore the proposed approach to control and manage requirements changing risks in IT software projects. The main categories and the corresponding factors are defined as:

A. Organizational Factors:

- Government regulations and politics.
- Business policy that is related to organization strategy and vision is changed.
- Business value of the system is decreased.
- Changing in organization structure.
- Changing in leadership and shifting business focus.
- Market competitors changed or increased.
- Decision-making procedures.

B. Project Factors:

- Software project domain, scope and role are unclear.
- Project size is large-scaled and requirements are overload.
- Project price/developing time are changed because of under or over-estimation.

C. Development Process Factors:

- Software product quality factors are changed or enhanced.
- Requirements priorities are not well-handled.
- Software product constraints are changed and have conflicts.
- Requirements priorities are changed.
- Complexity of development technology.

- Design weakness and the inter-relationships and interfaces among system components are complex.
- Changing the data structure.
- Large-volume of data is transformed into new technology.
- Redundant and irrationalized data objects.
- Misunderstanding of the system.
- Lack of using software tools.
- Limited usage of good programming styles.
- The associated documentations are not available or inadequate.
- Unclear feedback from design reviewers and /or requirement specification reviews.
- Uncertainty of upgrading the system platform.

D. Project Stakeholder Factors:

- Potential changes of users/customers/business managers demands.
- Developers experience and skills are limited and inadequate.
- Customers are not well-collaborated or may miss some requirements.
- Customers/users changed their minds and scope.
- Leadership characteristics and team member characteristics.
- Communication skills among development team members, project and organizational managers, customers and users.
- Strong appearance of market competitors.

At pre-processing stage, the studying of all the above factors and examining the interrelationships among them will reduce risks of system failure and help to make a decision on which changes to implement. To measure requirements volatility, the requirements functionalities, properties, dependencies, interrelationships, the main causes and issues are identified before suggesting the methodology. Changes management and risks control will help to enhance design, implementation, maintenance and the overall software quality. Fig. 1 depicts the main causes and the issues of controlling and managing the risks.

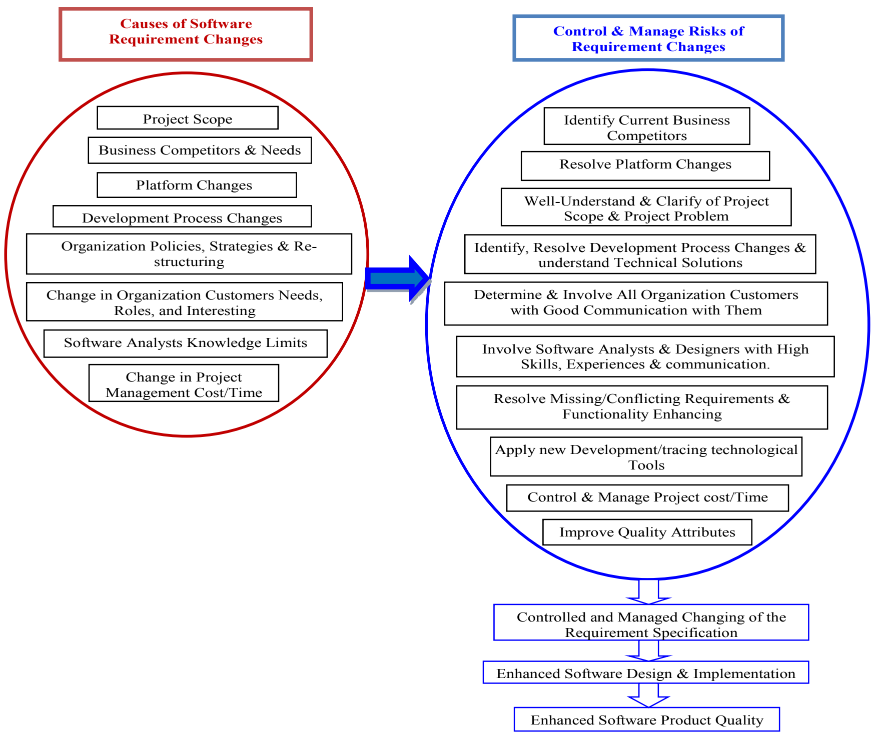
Causes of Software Requirement Changes
Project Scope
Business Competitors & Needs
Platform Changes
Development Process Changes
Organization Policies, Strategies & Re-structuring
Change in Organization Customers Needs, Roles, and Interesting
Software Analysts Knowledge Limits
Change in Project Management Cost/Time
Control & Manage Risks of Requirement Changes
Identify Current Business Competitors
Resolve Platform Changes
Well-Understand & Clarify of Project Scope & Project Problem
Identify, Resolve Development Process Changes & understand Technical Solutions
Determine & Involve All Organization Customers with Good Communication with Them
Involve Software Analysts & Designers with High Skills, Experiences & communication.
Resolve Missing/Conflicting Requirements & Functionality Enhancing
Apply new Development/tracing technological Tools
Control & Manage Project cost/Time
Improve Quality Attributes
Controlled and Managed Changing of the Requirement Specification
Enhanced Software Design & Implementation
Enhanced Software Product Quality

Figure 1. Main causes of requirements changes and issues to control and manage their risks

4. Methodology of Controlling Changing Risks and Managing Requirement Volatility

Dealing with reliable, complete, consistent and high-quality requirements is not easy task, it demands much effort from developers and maintainers, better involvement of interested stakeholders and a solid knowledge in quality management techniques. The suggested comprehensive methodology focuses on the formal tasks that are needed to be followed to change requirements particularly in the late stages of software development, and after completing requirement specification in requirement engineering phase. The objectives of using this model are to better understand the change causes and their underlying failure risks; during the later stages of software development process, understand the requirement volatility process, enhance decision-making process, improve the productivity of implementing change requirements, reduce failure risks and enhance the software product release quality.

The proposed model is based on descriptive and qualitative methods. Descriptive analysis provides rich information to understand the corresponding problems of requirements volatility as well as related aspects; such as organizational policy, customer needs and product changes, whereas qualitative methods are employed to analyze the collected data and to evaluate the change process. The ad-hoc maturity approach has a set of interrelated actions and activities to manage requirement volatility, and to control and reduce risks impacts. The work information model is represented in Figure 2.

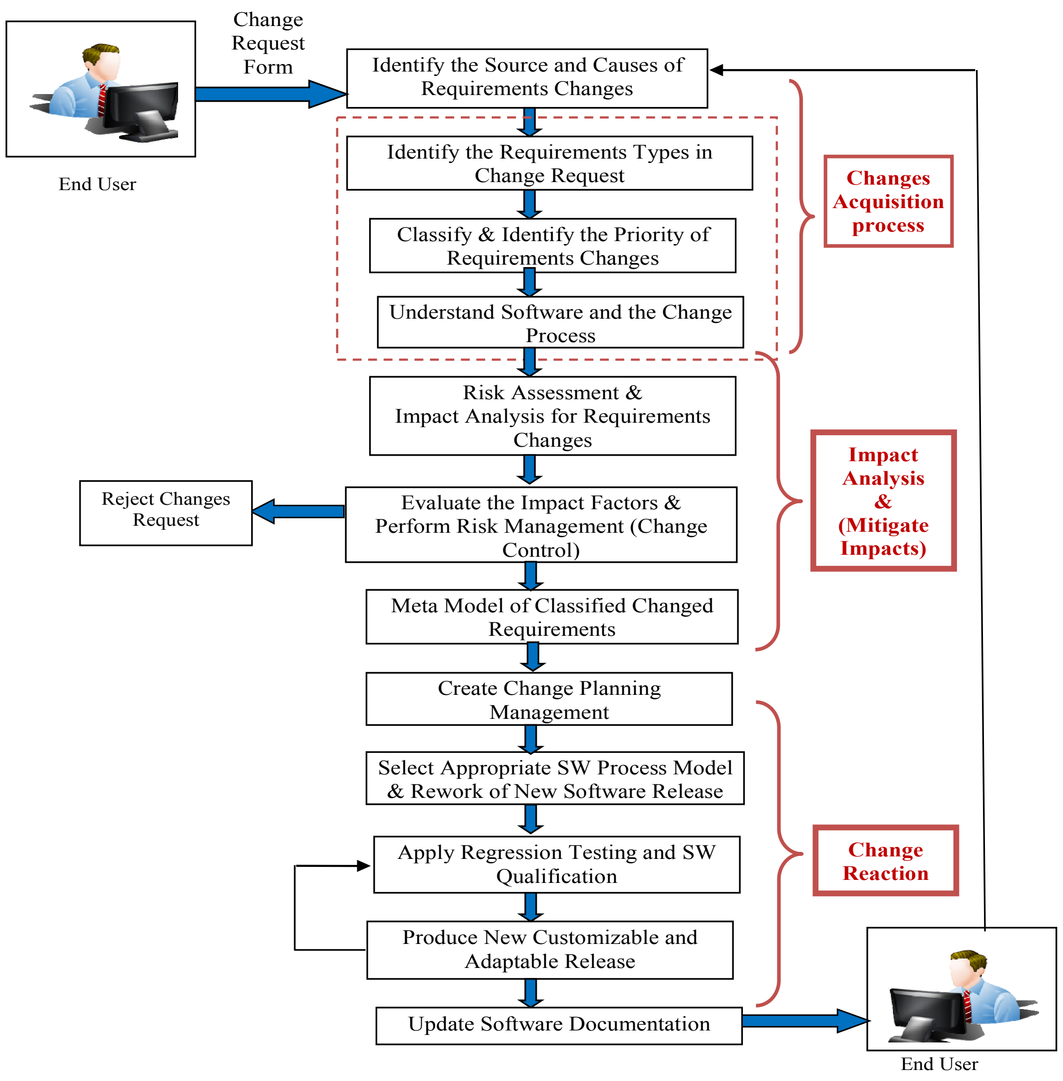
Change Request Form
End User
Identify the Source and Causes of Requirements Changes
Identify the Requirements Types in Change Request
Classify & Identify the Priority of Requirements Changes
Understand Software and the Change Process
Changes Acquisition process
Risk Assessment & Impact Analysis for Requirements Changes
Reject Changes Request
Evaluate the Impact Factors & Perform Risk Management (Change Control)
Meta Model of Classified Changed Requirements
Impact Analysis & (Mitigate Impacts)
Create Change Planning Management
Select Appropriate SW Process Model & Rework of New Software Release
Apply Regression Testing and SW Qualification
Produce New Customizable and Adaptable Release
Update Software Documentation
Change Reaction
End User

Figure 2. The work information model for requirement changes and impacts

4.1 Identify the Source and Causes of Requirements Changes

Change request is issued by customers, end users, business manager or according governmental regulation. The causes are explained previously in section 3. The formal request is written using a dedicated change request form. The form is checked and confirmed, then it is used through impact analysis. 4.2 Identify the Requirements Types in Change Request; whether the changes are in functional, data requirements, system platform, or in software project constraints. The customers, managers, technical staffs and end users are cooperated to identify the desired requirements and constraints that they really want.

4.2 Identify the Requirements Types in Change Request

Whether the changes are in functional, data requirements, system platform, or in software project constraints. The customers, managers, technical staffs and end users are cooperated to identify the desired requirements and constraints that they really want.

4.3 Classify and Identify the Subjective Priority of Requirements Changes

The priority is concerned to its relative importance in business or as suggested by business top managers or customers with high-position. The requirements have different importance and the customer may have different views about their importance. It is impossible to implement all requirement changes within limited time and budget during the current release. Priorities are identified according to user/customer preferences, the changes and business goals. These facts make requirements prioritization a critical task, especially when there are a large number of changes and have incompatible priorities. Therefore, the collaboration of the business managers, customers and users is needed to resolve this issue. It can be performed by:

- Prioritization of changes importance; where some changes are more critical than others, and
- Prioritization of development order; where some changes have great values to business and software capabilities.

Subjective prioritization is done iteratively and incrementally using Business Case Analysis technique, and using Technical Weighting Scale of range 1 to 5 that represents (low, low-moderate, moderate, high-moderate, high) correspondingly. It depends on weighting the importance to customers and business. Subjective prioritization is important to eliminate the impact of the requirement volatility, improve business value, and improve customers' satisfaction.

4.4 Understand Software and the Change Process

The developers need to well-understand the software development process model, software design, software functionalities, data and product documentation before proceeding with the changing procedures. Product documentations incorporate requirement specification, design documentation, source code of its programs and test oracle, if this documentations are not up-to-date it will make software maintenance difficult, but practically they were not updated after maintenance and are outdated.

4.5 Risk Assessment and Impact Analysis for Requirements Changes

It identifies the risk factors that are involved in implementing requirements change and examines during impact analysis of requirements changes how these factors are related to each other. Sometimes, the changing of one requirement may have impact on other dependent requirements and the changes may expand to affect organizational goals. Hence, it is used to assess which parts will change and which dependent parts are influenced with the changes. The changing propagation beyond the intended parts and examining methodically the existing environment to predict the potential effects make the evaluation of impact ambiguous and complicated task. The results of impact analysis need to be accurate to positively direct the changes control and management (Rahman, Razali R. & Singh, 2014) and to avoid failures in such projects. Therefore, the following issues are taken into our consideration in this study:

- The volume of incompleteness and inconsistent of requirements in changing request form.
- Requirement changing types; Requirement addition, modification, deletion.
- Requirement changes state (urgent or not).
- Changing priority and business values.
- Suitable selection of elicitation technique.
- Requirement dependencies; requirements functionalities and their interrelationships
- The complexity of changes.

- Software application domain, type and objectives.
- Software project size and age.
- Availability of up-to-date documentation (i.e. requirement specification, design specification, test suites).
- Software architecture style (i.e. data-centered, data-flow, object-oriented, or layered architecture), architectural patterns, factoring levels, and design models.
- Availability of source code of the programming language and the clarity degree of programming styles.
- Software development process model and implementation of design methodology.
- The degree of desired quality attributes.
- Organization polices and regulations
- Development platform (i.e. memory, CPU, communication and operating systems); for the post release requirements changes.
- Software engineers motivations, experiences and skills levels.
- Software engineering productivity rate.
- Customers and users' involvement and cooperation with software development team and relationship level between software developers and customers and frequent communication between them in requirement engineering phase and the following phases.
- Organization size and policies.
- Availability and usability of software tools.
- Underestimating or overestimating of Cost/Time/Effort in planning may negatively influence planning process.

The success of software development / maintenance projects depends on the investigation of risks, level of awareness concerning requirement volatility, the software project attributes, the management approaches used and the accuracy and correctness of impact analysis.

4.6 Evaluate the Impact Factors and Do Risk Management (Change Control Board)

The qualitative and quantitative methods are used to evaluate different impacts of changing requirements to determine the feasibility of implementing the requested changes. The evaluation that deals with assessing risks is performed by analysts, who eliminate the risks and their impacts.

The requirement volatility measures are used to determine the adequacy of software engineering process, resources, and developers' number and productivity. Requirement volatility measurement is important to find whether our project is moving on track or not, to know which changes have more impact on project than other one. It is also helpful to predict the future product and. Requirements volatility is measured using mathematical formula (Stark, Oman, Skillicorn & Ameele, 1999), (Roedler & Rhodes, 2010):

$$R_V \approx C / Q \quad (1)$$

Where:

R_V: Requirement volatility,

C: Cost, and

Q: Release Quality.

$$R_V = \frac{R_a + R_c + R_d}{VCN} \times 100 \quad (2)$$

Where:

R_v: Requirement volatility,

R_a: Added Requirements,

R_c: Modified Requirements,

R_d: Deleted Requirements, and

VCN: Version Content Notice = set of requirements agreed by both the developer and customer.

Requirements volatility can be measured depending on the size of change in use case model and is measured by counting the number of minor change, and major changes. It is defined as:

$$R_V = \frac{\sum_{i=1}^{m} changed\ requirements}{\sum_{J=1}^{n} Requirements} \tag{3}$$

Where:

R_v: Requirement volatility,

m: Total number of change requirements change (i.e. requirements addition, requirements deletion, and requirements modification), and

n: Total number of requirements for a certain development phase.

But Eq.1, Eq.2 and Eq.3; did not consider the increasing effect of requirement addition, changing or deletion in the late phases of software development when it is evolved and grown. Therefore, to find the influence of changing requirements at the late stages of software development on the requirement volatility, requirements volatility is defined as:

$$R_v = \frac{R_a + R_c + R_d}{VCN} \times (10^{p/2} - 1) \tag{4}$$

Where:

R_v: Requirement volatility,

R_a: Added Requirements,

R_c: Modified Requirements,

R_d: Deleted Requirements,

VCN: Set of requirements agreed by the developer and customer,

p: Number of software development passed phases, and

$(10^{p/2}-1)$: The volatility impact factor.

During the requirements engineering phase, p=0; thus $(10)^{0/2} = 1$, and $(10^{p/2}-1) = 1-1= 0$. It means that the requirement engineering phase has no considerable effect on volatility impact factor.

Requirement inspection and validation are also needed to discover requirement defects to enhance the requirements quality and succeed software projects. The square matrix is used as an evaluation process model to measure the quality of requirements changes, which will influence later the quality of new software release. Evaluation consists of five tasks:

a. Establishing requirements evaluation. This is performed through identifying the software type, preferred quality characteristics, and identifying the evaluation purpose and model?
b. Specifying requirements evaluation. It is done through identifying the internal and external quality issues and rating the each quality issues ranking. Correctness weight must be not less than 0.9 Completeness must be no less than 0.8 and consistency must be not less than 0.75.
c. Designing requirements evaluation.
d. Executing requirements evaluation.
e. Assess the results of measuring and comparing the criteria that are verified by experts.

4.7 Comprehensive Meta Model of Classified Changed Requirements

Semi-formal definitions of requirements that are classified into clusters related to requirement types (i.e. functional, data, and non-functional requirements); which in turn can be subdivided into subclasses and so on, the levels of details, implementation complexity, source of the requirements, requirement values, description of change, date of elicitation, and risk level (i.e. high or low risks), where:

Domain of Changed Requirement ⊆ Domain Changed Requirement Meta-Data.

Changed Requirement ⊆ Requirement Types x Composition x Levels of Details x Implementation Complexity x Source of the Requirements x Requirement Values x Description of Change x Date of Elicitation x Risk Level.

4.8 Create Change Planning Management

It is very important for project manager to have adequate information and knowledge about requirements changes introduced during software development to make the appropriate decisions. Since unstable requirements influence and exceed the planned software development time schedule and budget, the new planning estimation is needed; for the time schedule, cost and number skilled developers, according to new long-term requirements. After analyzing the possible consequences of changes on other software development phases and parts of the project, the plans are put to determine the assumed actions with respect to the predefined changes. Re-planning will assist to make reasonable decisions during the next development phases.

4.9 Select Appropriate SW Process Model & Rework of New Software Release

Selecting the software development model and selecting the appropriate requirement elicitation technique have an influence on the level of requirement specification details and product success rate (Rehman, Khan & Riaz, 2013). Implementation of requirement changes should match with the current software architecture and design, data design, and code. The modification of the internal structure of the software may be done without changing the functionality of the programming using software refactoring technique. Architectural refactoring is mainly needed to detect and remove design defects, which in turn will reduce the complexity, ensure the improvement of software quality and minimize the long-term maintenance. Verification is done after refactoring to ensure that the modules are functionally conformed to requirement specification and interact as designed. The test cases are redesigned to overcome the new and modified requirements. Integration testing is also applied to ensure that the modules are well-integrated to form the complete software product. The effort in implementing requirements changes implementation is high in the requirement specification, impact analysis, and testing phases.

4.10 Apply Software Qualification and Regression Testing for the Pre and Post-Release of System

Concerning requirement changes, with adequate test cases that cover the portions of software that are affected by changes. It is considered as an objective evaluation process that is used to determine the scope of the changing impact, monitor and detect the defects as a consequence of changes, and to verify software specifications to achieve its goals. To achieve the model objectives, test cases are then written to check on the functional validation and effectiveness. The model evaluation process is based on cost-benefit evaluation model.

4.11 Produce New Customizable and Adaptable Software Release

A new low-risk software release is issued after applying the customer's required changes and a matching between the expected software outcomes and the actual implementation outcomes.

4.12 Update the Related Software Documentation to fit the Applied Changes, Since Documentation Engineering is a Sub-Domain of the Software Engineering

It is needed to update the descriptive information that specifies the applied changes to be more consistent, well-understandable and up-to-date. Documentation is the basis for communication among development team members and a system information repository for maintainers. Documentation consists of many different kinds of documents produced at different points after changing requirements are performed.

5. Discussion and Key Findings

The study includes 10 different projects, and the changes are classified into added, modified and deleted requirements. The table 1 shows the acquired data, total changed requirements, total project requirements, requirements volatility rate and volatility impact factor. Figure 3 and shows the requirement volatility impact factor in design phase and figure 4 shows the requirement volatility impact factor in maintenance phase for the same 10 projects. During the study, sensitive data is collected to avoid as much as possible the negative effects of changing requirements and to perform risk management. .But in actual software applications, it is not an easy task, due to:

- The application end-users are not well-specified the change request forms (CRF). They did not explain the underlying reasons for the proposed change. Sometimes, CRF requires an experienced user.
- The requirements impact analysis (i.e. dependency analysis) of requirements change is not adequately performed and the maintainer is unable to predict the potential risks with requirements evolving.
- The complex relationships among requirements cause the ripple effects of requirement changes and management.
- Classifying requirement dependencies into different classes and identifying requirement dependencies and their relationship types are difficult tasks.

- The traceability between the changing requirements and other requirements and artifacts are not identified.
- Increasing in requirement volatility and their business values will raise the developer and maintainer job size, due to the additional work, and consequently increase the estimated project's cost, resources costs and time schedule.
- The application complexity
- Reconfiguration of reused set of requirements and software components.
- The collaboration among the developer, managers and the users will reduce cost overruns.
- The large number of participants and roles may influence negatively on the project development/maintenance work progress and quality, when adding new functionalities or classifying defects in requirements specification.

Table 1. Changed requirements and their impacts

Type of Changed Requirements	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6	Project 7	Project 8	Project 9	Project 10
Added Requirements	3	2	3	2	3	1	3	2	4	3
Delete Requirements	2	2	3	0	1	2	2	2	1	2
Modify Requirements	2	5	5	5	6	2	6	5	3	6
Total Number of changed Requirements	7	9	11	7	10	5	11	9	8	11
Total Number of Requirements	65	74	60	83	67	58	79	63	59	71
Change Percentage %	10.77	12.16	18.33	8.43	14.93	8.62	13.92	14.29	13.56	15.49
Requirements Volatility Rate	0.11	0.12	0.18	0.08	0.15	0.09	0.14	0.14	0.14	0.15
Volatility Impact Factor	0.97	1.09	1.65	0.76	1.34	0.78	1.25	1.29	1.22	1.39

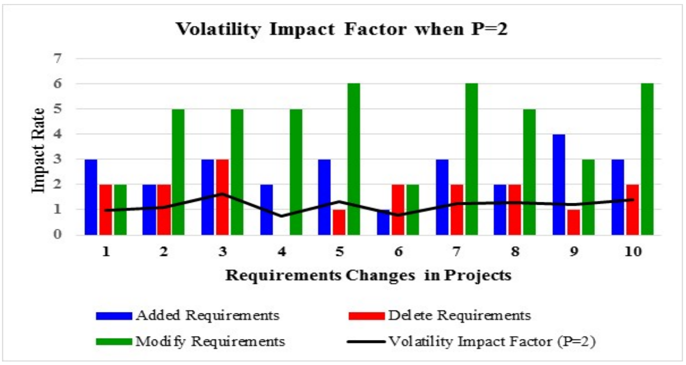

Figure 3. Volatility Impact Factor when P=2 (Design)

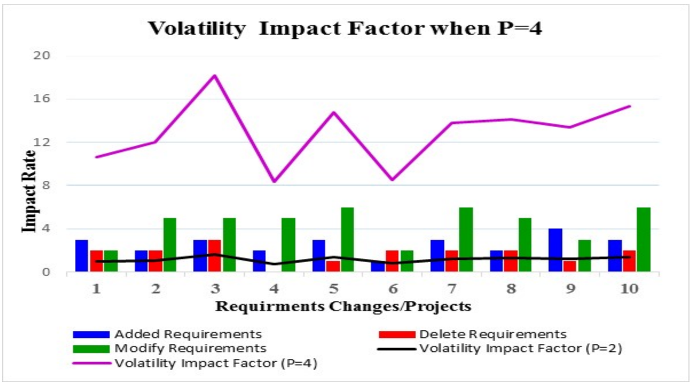

Figure 4. Volatility Impact Factor when P=4 (Maintenance)

6. Conclusion

Software developers need high-level of expertise and knowledge to deliver high quality software products, which increase the requirement engineers' efforts to elicit and analyze the requirements, represent them into comprehensive models and process in systematic and qualitative technique. Requirements volatility has an influence on different phases of SDLC and software maintenance; hence it has a great impact on the estimated project cost, schedule and the quality of final product release. In this paper, we have studied different aspects of requirements volatility, and analyzed the reasons for requirements changes, their risks, and the impact analysis on software development process.

Requirements volatility cannot be fully avoided but its impact can be reduced and its causes can be minimized. A requirement volatility model is presented, which included the information required for the change acquisition process, impact analysis by measuring the size of changed requirements in use case models and classifying them into major and minor changes, to find the influence of changing requirements at the late stages of software development. The planning management is measured accordingly to produce new customizable and adaptable release. The model will help to reduce the influence of changing requirements on the design or implementation of the software. If the model is not adopted in a right way then the developers may face more problems; depending on the software application type and size. The impact of requirement change is used to evaluate their risks and to understand the degree of risk. If the requirement volatility rate is high, after completing requirement specification; then the scheduled time, costs and efforts increase and may lower release quality.

Acknowledgment

The authors are grateful to the applied science private university, Amman-Jordan, for the full financial support granted to cover the publication fee of this research article.

References

Bhatti, M. W., Hayat, F., Ehsan N., Ahmed, S., Ishaque, A., & Z.Sarwar, S. (2010, Oct. 8-10). An investigation of changing requirements with respect to development phases of a software project. *International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, 323-3327, Oct. 8-1, 2010, Krakow, Poland.

Damian, D., Chisan, J., & Thamsamy, L. V. (2005). Requirements Engineering and Downstream Software Development: Finding from a Case Study. *Empirical Software Engineering, 10*(3), 255-283. http://dx.doi.org/10.1007/s10664-005-1288-4

Dev, H., & Awasthi, R. (2012). A Systematic Study of Requirement Volatility during Software Development Process. *International Journal of Computer Science Issues (IJCSI), 9*(2), 1, March.

Ferreira, S., Collofello, J., Shunk, D., & Mackulak, G. (2009, Oct.). Understanding the Effects of Requirements Volatility in Software Engineering by Using Analytical Modeling and Software Process Simulation. *Journal of Systems and Software, 82*(10), 1568–1577. http://dx.doi.org/10.1016/j.jss.2009.03.014

Kulk, G. P., & Verhoef, C. (2008). Quantifying requirements volatility effects. *Science of Computer Programming, 72*(3), 136–175. Elsevier. http://dx.doi.org/10.1016/j.scico.2008.04.003

Loconsole, A. (2008, June 26-27). A Correlational Study on Four Measures of Requirements Volatility. *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, University of Bari, Italy. Retrieved from http://www.bcs.org/upload/pdf/ewic_ea08_paper18_1.pdf

McGee, S., & Greer, D. (2012). Towards an Understanding of the Causes and Effects of Software Requirements Change: Two Case Studies. *Requirement Engineering*, Springer-Verlag, London, *17*(2), 133-155. http://dx.doi.org/10.1007/s00766-012-0149-0

NIST (National Institute of Standards and Technology). (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing*. Gaithersburg, Maryland, RTI Project No. 7007.011, U.S Department of Commerce Technology Administration. Retrieved from http://www.nist.gov/director/planning/upload/report02-3.pdf

Nurmuliani, N., Zowghi, D., & Powell, S. (2004). *Analysis of Requirements Volatility during Software Development Life Cycle*. In Proceedings Australian software engineering conference, ASWEC'04, Australia, 28–37. http://dx.doi.org/10.1109/ASWEC.2004.1290455

O'Neal, J. S. (2001, November). *Analyzing the Impact of Changing Requirements*. 17th IEEE International Conference on Software Maintenance (ICSM'01), 190-195. http://dx.doi.org/10.1109/ICSM.2001.972729

O'Neal, J. S. (2003). *Analyzing the Impact of Changing Software Requirements: A Traceability-Based Methodology*. Department of Computer Science, Faculty of the Louisiana State University and Agricultural and Mechanical College (Doctoral dissertation). Retrieved from http://etd.lsu.edu/docs/available/etd-0929103-120652/unrestricted/O'Neal_dis.pdf

Orso, A., Apiwattanapong, T., Law, J., Rothermel, G., & Harrold, M. J. (2004, 23-28 May). *An empirical comparison of dynamic impact analysis algorithms*. Proceedings of the 26th International Conference on Software Engineering (ICSE'04), Edinburgh, Scotland, 491–500. http://dx.doi.org/10.1109/ICSE.2004.1317471

Rahman, M. A., Razali, R., & Singh, D. (2014, Jan.). A Risk Model of Requirements Change Impact Analysis. *Journal of Software, 9*(1), 76-81. http://dx.doi.org/ 10.4304/jsw.9.1.76-81

Rehman, T., Khan, M. N. A., & Riaz, N. (2013 February). Analysis of Requirement Engineering Processes, Tools/Techniques and Methodologies, *International Journal of Information Technology and Computer Science, 5*(3), 40-48. http://dx.doi.org/10.5815/ijitcs.2013.03.05

Roedler, G., & Rhodes, D. (2010, Jan. 29). *Systems Engineering Leading Indicators Guide: Version 2*. Massachusetts, Institute of Technology, INCOSE, and PSM. Retrieved from http://seari.mit.edu/documents/SELI-Guide-Rev2.pdf

Singh, M. P., & Vyas, R. (2012, Sep.). Requirements Volatility in Software Development Process. *International Journal of Soft Computing and Engineering (IJSCE), 2*(4), 259-264. Retrieved from http://www.ijsce.org/attachments/File/v2i4/D0960082412.pdf

Stark, G., Oman, P., Skillicorn, A., & Ameele, R. (1999). An Examination of the Effects of Requirements Changes on Software Maintenance Releases. *Journal of Software Maintenance: Research and Practice, 11*(5), September/October. 1999, 293-309. http://dx.doi.org/ 10.1002/(SICI)1096-908X

Sudhakar, M. (2005, Apr.). *Managing the Impact of Requirements Volatility*, Department of Computing Science, Umeå University, Sweden, 29th April (Master MSc. Thesis). Retrieved from http://www8.cs.umu.se/education/examina/Rapporter/MundlamuriSudhakar.pdf

Thakurta, R. (2010). Management of Requirement Volatility - A Study of Organizational Competency and How It Is Influenced by The Project Environment. *Journal of Information Technology Management, XXI* (2), 24-34. Retrieved from http://jitm.ubalt.edu/XXI-2/article3.pdf

Thakurta, R., & Ahlemann, F. (2010). *Understanding Requirements Volatility in Software Projects: An Empirical Investigation of Volatility Awareness, Management Approaches and Their Applicability.* Proceedings of the 43rd Hawaii International Conference on System Sciences. http://dx.doi.org/ 10.1109/HICSS.2010.420

Tripathy, P. I., & Naik, K. (2015). *Software Evolution and Maintenance: A Practitioner's Approach*. John Wiley & Sons, Jan 2015.

Zhao, J., Yang, H., Xiang, L., & Xu, B. (2002). Change Impact Analysis to Support Architectural Evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 317-333. http://dx.doi.org/10.1002/smr.258