Managing the Impact of Requirements Volatility

Master Thesis, 2005

Mundlamuri Sudhakar

Department of Computing Science
Umeå University
SE-90187 Umeå, Sweden

Thesis Defense: April 29th 2005, MIT building.

Thesis Supervisor: Annabella Loconsole, Department of Computer Science, Umeå University, Sweden.

Contents

List of Tables and Figures:

Acknowledgement

I would like to thank Annabella Loconsole for giving me support to complete my thesis successfully in time by giving suggestions, reviewing my thesis and providing me good articles.

My special thanks to Mr Per Lindström to help me in my thesis work and for my entire studies in Umeå university with friendly helping nature.

I would like to thank my parents and friends for giving external support to complete my thesis.

Finally, I would like to thank to Umeå University for giving me the opportunity to study here.

Abstract

Software requirements are very important in software development. The success of software project depends on the quality of requirements specification. Even though we have good requirements specification in the beginning, there will be requirements changes during the project development.

Requirements change during the system development is also known as Requirements Volatility. Requirements volatility has great impact on the cost, the schedule and the quality of final product. Due to this requirements volatility many projects are fail and some are completed partially. Requirements volatility can not be overcome fully but we can minimize some causes.

In this thesis I describe the causes of requirements volatility. How these causes will affect the different phases of software development life cycle. I studied and analyse the software development models and different methods regarding the requirements volatility and describe advantages and disadvantages of using different software models. Finally, I suggest how to Manage and control requirements volatility in different software development phases.

Chapter 1

Introduction

1.1 Background

Requirements collection is an early phase of the software development life cycle. Requirements are the foundation of the software development. They provide the basis for cost estimation and for planning project schedules as well as for designing and testing specifications. The success of software product, both functionally and financially is directly related to the quality of the requirements. Although the initial sets of requirements are well documented, requirements changes will occur during the software development lifecycle. Thus constant changes (addition, deletion, modification) in requirements during the development lifecycle impact the cost, the schedule and the quality of final product.

Lamsweerde [2], conducted a survey over 8000 projects from 350 US companies and revealed that one third of the projects are never completed, and one half succeed partially. In his survey he found that the requirements changes had 11% in total causes of failure to software projects. Many projects are discarded in the coding phase, and some are discarded in the designing phase because of requirements volatility. The Requirements volatility is occurring because of poor requirements collection activities, lack of experience, external factors, and many other reasons [see section 2.2].

In today’s world of rapidly changing the business and technology requirement, this is something that is inherent in the software development process and can not be ignored. These requirements changes are occurring during the development process. If a project has to be successful, requirements changes need to be managed.

Software development is a dynamic process therefore we can not avoid the requirements volatility. The problem is not with changing requirements; the problem is with prediction of requirements volatility and inadequate approaches for dealing with requirements volatility.

1.2 Goals of the thesis

The main goal of this thesis is to investigate different aspects of the Requirements Volatility (RV).

- What are the causes of volatility?
- Different types of metrics for requirements volatility. How it affects the project development schedule, project performance and quality of the product.
- Describe different ways of reporting volatility; what the different causes are and how much it affects on the success of a project.
- Investigate the different software development models in relation to the requirements volatility.
- How to manage requirements volatility in different phases.

1.3 Outline of the thesis

Section 1: Brief introduction of the thesis. It describes the goals and contents of thesis.

Section 2: Definition, causes and metrics of requirements volatility.

Section 3: Requirements volatility impacts on different phases of software development cycle and different factors like project schedule, project cost and quality of product.

Section 4: How can we manage and control requirements volatility in different phases of software development.

Section 6: Summary and conclusions of thesis.

Chapter 2

Requirements Volatility

Requirements volatility often results in significant growth in requirements size from the time of initial requirements specification to final requirements of the system development. Requirements volatility is an important risk in software project success, that can occur in multiple points during the software development process. These changes take place "while the requirements are elicited, analyzed and validated and after the system has gone into service" [7], simply through the software development lifecycle.

The requirements volatility has great impact for the success of software development. The chaos report says that the requirements volatility has 11% in total percentage causes in failures of software project [10]. Requirements volatility will affect the project schedule and cost overrun, project performance, and project quality. In fact every phase of software development is effected by requirements volatility.

Software development is considered to be a dynamic process where demands for changes seem to be inevitable [23]. Therefore the problem is not with requirements volatility, the problem is with inadequate approaches for dealing with them in a way that minimizes and communicates the impact to all stakeholders.

2.1 Definition of Requirement

Requirements describe the system's behavior. Every software development process starts with the requirements collection. The customer has some notation of what the system will do. We collect the customer needs for a software system and prepare the requirements documents according to the customer needs. Requirements document will give an understanding of the system, we can say a blue print of the system. A **requirement** is a feature of the system or a description of what the system is capable of doing in order to fulfill the system's purpose [18].

Requirements depend upon the project; they differ from project to project. Mainly we are using two types of requirements, functional and nonfunctional requirements. **Functional requirements** describe an interaction between the system and its environment. For example calculating total marks of student in all subjects, giving the grade and rank. **Nonfunctional requirements** describe a restriction of the system that limits our choices for constructing a solution to the problem. For example hardware and man power resources etc [18].

2.2 Changes to requirements and change requests

Once we collect the requirements from the customer, we sign the agreement with the customer. In the agreement both the parties agree what are all the requirements to be developed in the system. Theses requirements are called baseline requirements.

Any changes to the baseline will be done only through the submission of change requests. The following is a suggested process for requirements change management:

- The need for a change must be identified
- A change request is submitted.
- The change request must be approved or rejected.
- Approved changes are documented and incorporated into the baseline.
- Submission of a new baseline should involve a formal review.
- Submitted materials should contain sufficient details so that changes can be backed out if necessary.

All change requests will be reviewed on a regular basis by the project change control board. The change manager will drive the schedule based on the number and complexity of change requests and a Cost/Schedule Impact Analysis (CSIA) is demanded. The requirement change control team reviews the requested change and either accepts, reject, or defer and also ensure that the resources are neither scarce nor wasted during change activities [19].

2.3 Definition of Requirements Volatility

There is no standard definition of requirements volatility. Usually it expresses the changing nature of requirements over the system development life cycle.

Nidumolu [6], described the uncertainty of requirements as the difference in the information necessary to identify user's needs and information possessed by the developers.

Requirements volatility is defined as "the number of Requirements changes (addition, deletion, modification) in a specified interval time (week, month, year or in particular phase)" [7].

The change to requirements—after the basic set of requirements has been agreed by both the clients and developers of the requirements – are known as requirements volatility [4].

Javeed [1], have a definition of requirements volatility which depend upon the phase and stage it occurred. Requirements changes will occur from the first phase (System/Information Engineering and modeling) to the final phase (Maintenance) of the software development life cycle, depending upon the time (phase) it occurs. The changes can be described as pre and post release requirements changes.

1. Pre-release requirements changes:

a. Pre –FS (Functional Specification): Requirements changes which refer to changes to requirements during the early phases, elicitation, elaboration, analysis, modeling and negotiation of software development, before the functional specification has been completed and signed off.

b. Post –FS (Functional Specification): Requirements changes will occur during the later phases of software development lifecycle i.e. design, coding, development and testing, after the FS has been formally signed off by the client and product developers.

2. Post-release requirements changes:

The changes occur once the system has been deployed at the client side, after the system has been released. This occurs in maintenance phase.

In the above context, it is worth mentioning that the first type (1.a) of changes is constructive if correctly done, because these changes would help in defining more complete requirements. However the 1.b and second type of requirements can be destructive, as they affect the productivity in terms of cost overruns, schedule overruns and quality .Therefore, to keep track of Post-FS and post release changes to requirements is more important [1].

2.4 Causes of requirements volatility

Requirements evolution is due to internal and External Factors (social view point).

1. External Factors:
   a. Government regulations
   b. Market competitors
2. Internal factors
   1. In technical view point
      a. Product constraints
      b. Lack of experience to the project development team professionals
      c. Feed back from other phases of SDLC
      d. Project size/ requirements overload
      e. Software (price changes) and hardware.
   2. Potential for change (changes in business environment)
   3. Requirements in stability (the extent of fluctuation in user requirements)
   4. Requirements diversity (the extent to which stake holders disagree among themselves deciding on requirements).

5. Requirements analyzability (the extent to which the process of producing requirement -specification can be reduced to objective procedure).
6. Poor communication between users and development team.
7. Client irresponsibility

In 1994/95, the Standish group surveyed over 8000 software projects, and identified that the project failures is caused by poor requirements activities. More specifically: lack of user involvement (13%), requirements incompleteness (13%), change requirements (9%), unrealistic expectations (10%), and lack of planning (8%) [18].

Among the causes of requirements volatility, some can not be avoided, some other can be avoided by taking precautions. For instance, government regulations and market competitors can not be avoided. While we can minimize the requirements volatility effects of: communication between the customer and developer, client irresponsibility, feedback from other phases of software development life cycle.

2.5 Requirements volatility measures

Software measurements run through all phases of software development. Measurement is needed for assessing the status of projects, products, processes and resources. Software measurement makes possible to assess the software development process and in case something is wrong, it allows us making the adjustments to the process**.**

Measures can be used to collect the information about requirements volatility. Requirement volatility measurement is important to know which changes have more impact on project. It is also helpful to predict the future product and find whether our project is moving on track or not. Some measures of requirements volatility are:

- Number of changes (addition, deletion, modifications) classified by reasons for changes in a given time interval; cumulative number of changes, total number of requirements [5].
- Pre/post functional specification changes and post release changes [1].
- Preliminary design review, critical design review, post critical design review, average changed SLOC, average change modules, average change SLOC per module [3].
- Amount of information contained in requirements at a certain time [16].
- Requirements changes in time, additions, deletions, and modifications to software [17].

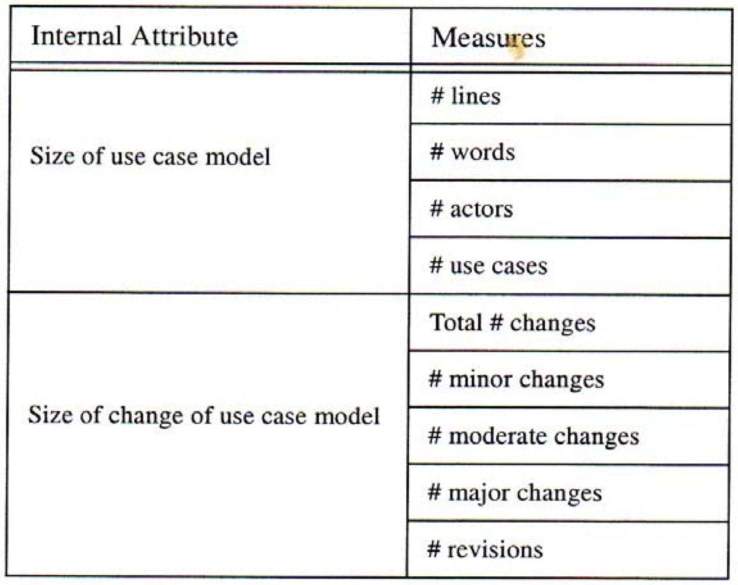
Measuring the Requirements volatility percent:

Stark [4], derived a formula based on the statistics on different projects:

Requirements volatility = (added + deleted + changed (modified))/ (# requirements in VCN)*100[4].

Where VCN (version content notice) = set of requirements agreed by both the developer and customer.

Loconsole [27] has conducted a case study on Requirements volatility measures for the long duration project of three years. She collected data in 14 use case models. She collected data for the measures minor changes, moderate changes, major changes, no lines, no of words, and more. Internal attributes for these measures are shown in table 2.1. As a result of the case study moderate changes were more present than minor and major changes. Requirements volatility is independent of number of revisions[1]. There is correlation between the size (lines of code and number of words) of use case module and the size of change. There is no significant correlation between the measures and the subjects rating volatility. There is significant relationship between the number of words and number of changes.



Internal Attribute	Measures
Size of use case model	# lines
	# words
	# actors
	# use cases
Size of change of use case model	Total # changes
	# minor changes
	# moderate changes
	# major changes
	# revisions

Table: 2.1 Internal Attributes and measures of volatility [27].

The percentage of requirements added, modified, or deleted depends on many things, like product size, duration of project, no of modules. Some examples (not all-inclusive) include what part of the life cycle you are in, what kind of product you are intending to build, the volatility of your intended market, the volatility of technology, and so forth.

1: Number of revision: in which time we are modifying the product according to new requirements.

We will grade the impact of requirements on project like high, medium, low. These will tell you the importance of requirement change. The reason you want to use volatility metrics is to give you advance warning that your project is off track. A better approach to using volatility metrics is how much requirements volatility is reasonable throughout the life of the project. For example, at the beginning of a project, when you are in the midst of discussing requirements issues, you would expect volatility to be higher for a healthy process.

Requirements volatility metrics provide the way to rapidly assess whether the degree of and reasons for requirements changes are consistent with current development activities. They consist of counts of requirements changed (added, deleted, modified) in each specification over a given time interval. Metric reports can be written in three types of information: graphical, tabular or numeric, and textual.

2.6 How to report requirements volatility

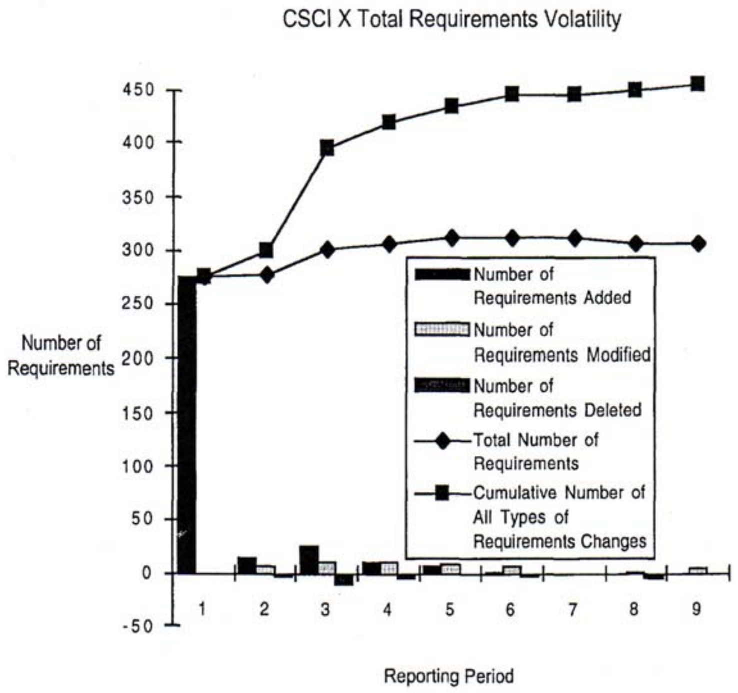


Figure 2.1: Sample metric report: total requirements volatility [5].

Fig: 2.1 shows one way to report requirements volatility [5]. The vertical bar shows the no of deletions, additions and modifications to the requirements in the specification over nine reporting periods. In the first reporting all actions are classified as "additions". The requirements changes line (Y-axis) shows the cumulative number of requirements changes from all reporting periods, and the requirements total line indicates the total number of requirements in the specification. Finally, requirements volatility will occurs in all reporting periods, but gradually decreases in regular intervals finally becomes very low.

Another way to report requirement volatility is to count the types of changes. There are different types for additions, deletions, and modifications.

In fig: 2.2 the dominant reason for requirement addition is changes in external interfaces and the second most frequent reason is oversight in an earlier version of specification. Customer requested enhancement will also play an important role in requirements volatility, while the others have a small impact [5].

In the two analyses presented requirements volatility is measured and graphed according to the additions, deletions, and modifications. In the first graph requirements volatility shown according to the reporting period. The second graph shows the additions, deletions, modifications according to the reasons which cause those changes.

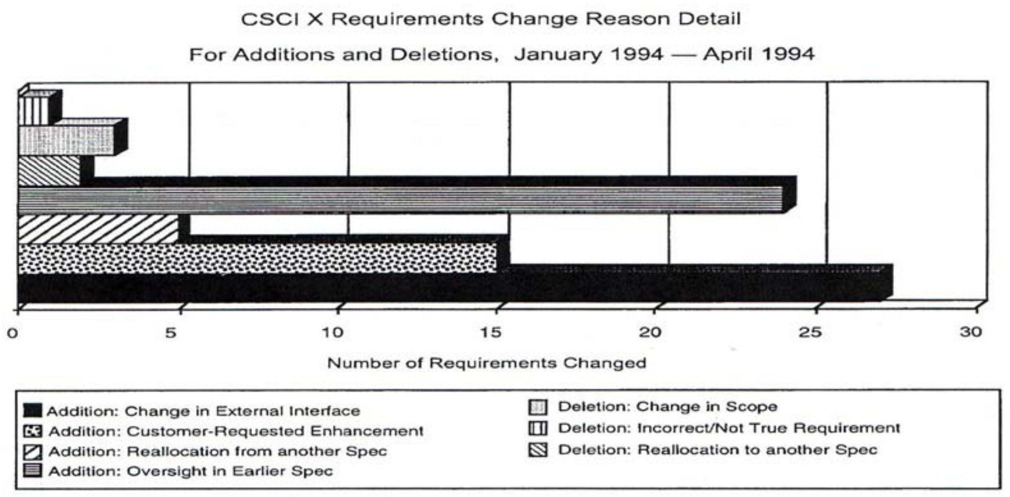


Figure 2.2: Sample metric report: requirements change reason [5].

Chapter 3

Impact of Requirements Volatility

Requirements volatility has a great impact on software development life cycle. The requirements volatility affects software releases and has major impact on project schedule, cost and project performance. The degree of requirements volatility is negatively associated with software project schedule and cost performance.

Lamsweerde conducted a survey, from the data collected from over 8000 projects from 350 companies in USA and revealed that one third of the projects were never completed, and half succeed only partially, i.e. with partial functionalities. Many projects have cost overruns and significant delays [2].

3.1 Project Schedule

Software development life cycle includes many steps like, design, development and testing. However, before committing for software development, the customer usually wants to know how much the project will cost, the time it will take, and what are all the internal activities and milestones.

A project schedule describes the software development cycle for a particular project by enumerating the phases or stages of project and braking each into discrete tasks or activities to be done. The schedule also portrays the interactions among these activities and estimates the time each task or activity will take. Thus the schedule is a time line where the activities will have start and an end, and when the products will be ready [18].

The project schedule is the first task when we develop a software product. Software cost is also depending upon the project schedule. Software project schedule is consisting of intermediate deliverables, design documents, application source code, meeting with stakeholders to collect the requirements, and to approve the intermediate work.

When there are requirement’s changes, these changes can affect design, code, and testing. The final effect of this requirements volatility can be missed deadlines. When you miss one deadline you need to postpone all subsequent deliverables deadlines. Sometimes you need to reschedule the total project schedule. A large number of software projects are stopped in the middle due to this reason.

Some times it is more important to develop a software project in the time scheduled rather than with high quality. For example, in banking sector when your competitor is introduced a new system offering the online banking facility you observed that your customers are moving to that bank attracted by new online banking facility and the new customers’ rate is decreasing gradually. Therefore you decide to introduce online banking facility. You can not wait long time for this project. You need to implement it as soon as possible with minimum requirements facilities. Otherwise for each day of delay, you will loose old and new customers, so you need to complete the project with minimum high priority requirements.

In these situations we can follow the method described in fig: 3.1 for prioritizing the requirements.

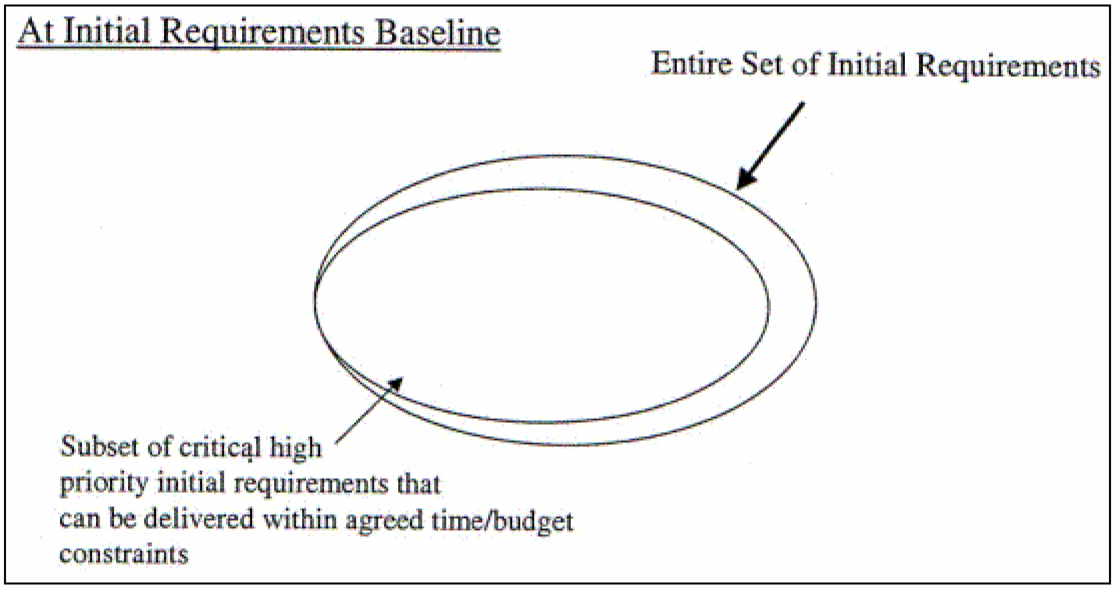

At Initial Requirements Baseline
Entire Set of Initial Requirements
Subset of critical high priority initial requirements that can be delivered within agreed time/budget constraints

Figure: 3.1. The high Priority requirements are a subset of the overall requirements [9].

When the developer works with the customers, the developer should identify the most critical, high priority requirements from the large requirements set. The size of the sub set requirements should be selected on time constraints (it means how much time we have to complete the project). The non–critical requirements will be either discarded or slated for the later development effort.

As new requirements are discovered, just as the initial requirements set some will have higher priority than others. In addition, new requirements will have higher priority than requirements in the existing baseline. When new requirements are discovered, but before they are accepted for development, the set of baseline requirements (initial requirements set at the time of agreement between the customer and the developers) will have to be adjusted to reflect the new prioritized requirements set.

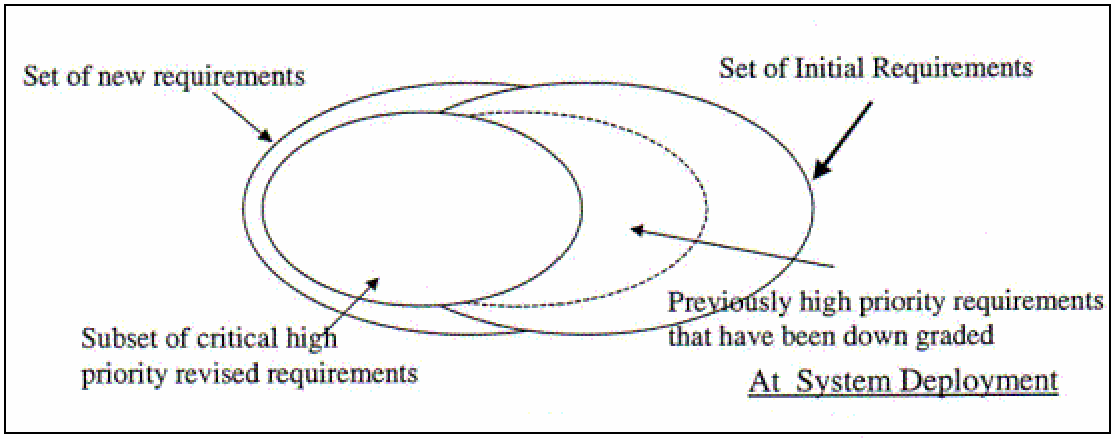

Set of new requirements
Set of Initial Requirements
Previously high priority requirements that have been down graded
Subset of critical high priority revised requirements
At System Deployment

Figure: 3.2. Requirements at System Development [9].

It is important to give the right priority to requirements, otherwise you may loose some critical business needs. This will result with an overworked development team, cost overruns, unhappy customers and missed business opportunities.

Stark [4] has derived the formula to find whether the project is completed in the specified time or not. Whether the project is completed before the deadline or on deadline or after the deadline. If the value of planed schedule percent is equal to 100 project was completed on dead line, if greater than 100 the project completed after the dead line, if the value is less than 100 the project was completed before the dead line.

Planed schedule percent = {1+((actual days - planed days)/(planed days))}*100

If the value is 100, it means the original plan was met (actual days = planed days)

If the value is greater than 100, the release was late (actual days > planed days)

If the value is less tan 100, the release was delivered early (actual days < plane days).

3.2 Project performance

Requirements volatility has impact on the performance of the development lifecycle. As we discussed earlier it has impact on every phase of software development life cycle.

The instability of requirements is characterized by the significant fluctuation of user's requirements in the later stages of the development. It is also characterized by the difference between requirements that were identified at the beginning of the project and requirements that existed at the end. The later aspect of requirements volatility is influenced by the differences and conflicts among users/stakeholders of requirements.

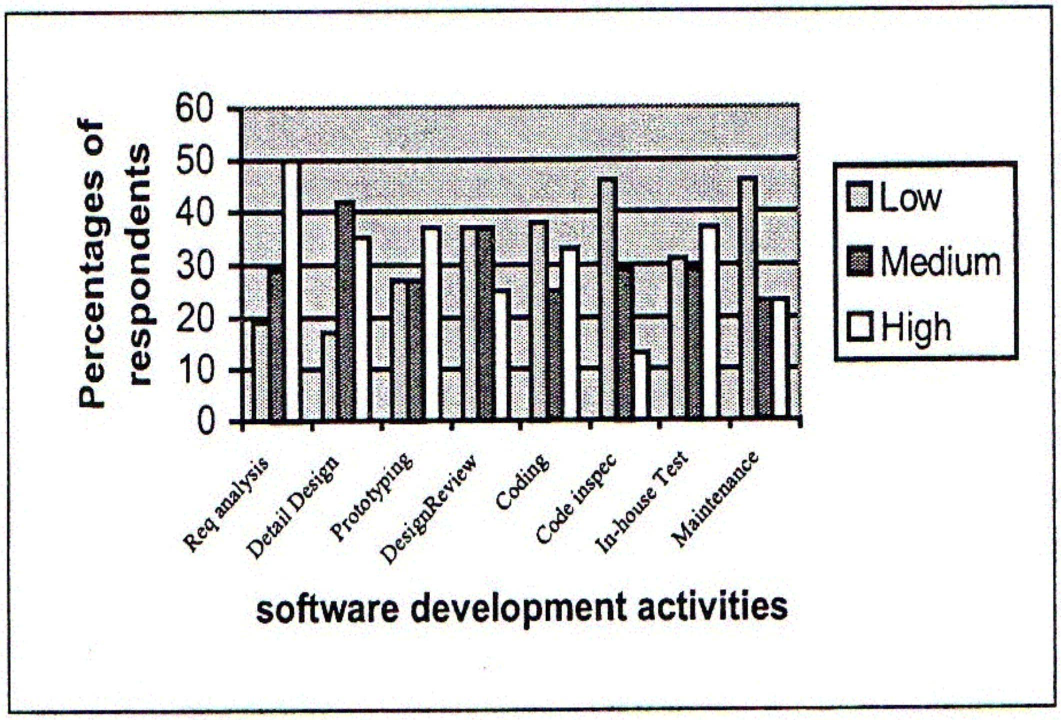


Figure: 3.3. The level of requirements volatility from the respondents' perspective [23].

In a study performed by Zowghi and Nurmuliani, the developers were asked to rate the level of requirements change for each stage of software development activity. The result in figure3.3 shows that most of respondents indicate that the level of changes during the requirements analysis is high. In the requirements analysis stage, software requirements are being explored, elaborated and fleshed out while the new requirements are being discovered as the result of the analysis. At the end of prototyping stage, not surprisingly the requirements fluctuated as the users change their mind after they were shown early prototypes.

The level of requirements volatility seems to decrease as the project progress and near to the end of life cycle, but rise slightly again during coding and maintenance-house testing.

3.3 Project cost

Software project cost estimation is an important factor for software project managers. There is no relationship between cost, effort, and pricing [11]. The price depends upon the market competitors. Some times the developers are ready to give fewer prices because they may intend to use this code in future related projects, and their employees will get experience on those areas of software development. When we talk about cost effort it depend mainly upon project duration, number of employees you need, (cost of employees), software and hardware efforts, building rents, traveling expenses and so on including requirements volatility. Based on the size of new requirements, we need to check which requirements will come under the baseline requirements and which are not.

For new requirements developers need to make an agreement with new revised cost and schedule of the project.

Re-planning according to new requirements:

When new requirements are received from the customers the project managers need to re-plan the schedule and the cost of the project. The developers need to negotiate with customers to make requirements tradeoffs, additions, and rejections. Developers could limit the functionality and defer the full functionality to the later releases.

Re-planning of these new requirements is important because of their dependency on other requirements. Rework of software that implements the existing requirements will cost more, based on the new requirements.

Requirements volatility plays important business logic when calculating the project cost. The managers will quote the low price for the first stage of development and will fix higher price for the additional requirements changes (additions, deletions, modifications). Here you need to remember that the price of the product is decided by parties, developers and stakeholders with the initial requirements specification before the developers start the project.

3.4 Software maintenance

Large and long term applications will have frequent requirements changes. In maintenance environment, new requirements are added to release or existing requirements are deleted or modified. This will affect cost, schedule and quality of the software project. As we update the previous versions the quality of product increases. Stark said that in 44 releases the requirements volatility was 48% [4].

In maintenance phase the requirements volatility is caused mainly by government regulations, market competitors, and changes agreement between stakeholders, changes in business environment and changes in the regulations in their partner companies.

How much volatility do our project release experience? What kind?

According to Stark, total of 123 requirements changes were made to the 44 software releases under study. Fig: 3.4 shows the releases that experienced requirements volatility. Sixteen of the deliveries (36%) had no requirements change; of these, seven were made according to schedule, five more were with in 15% of the original scheduled date, and five were more than 15% late. The remaining 28 releases (64%) were affected by requirements changes, with nine of them having greater than 50% volatility.

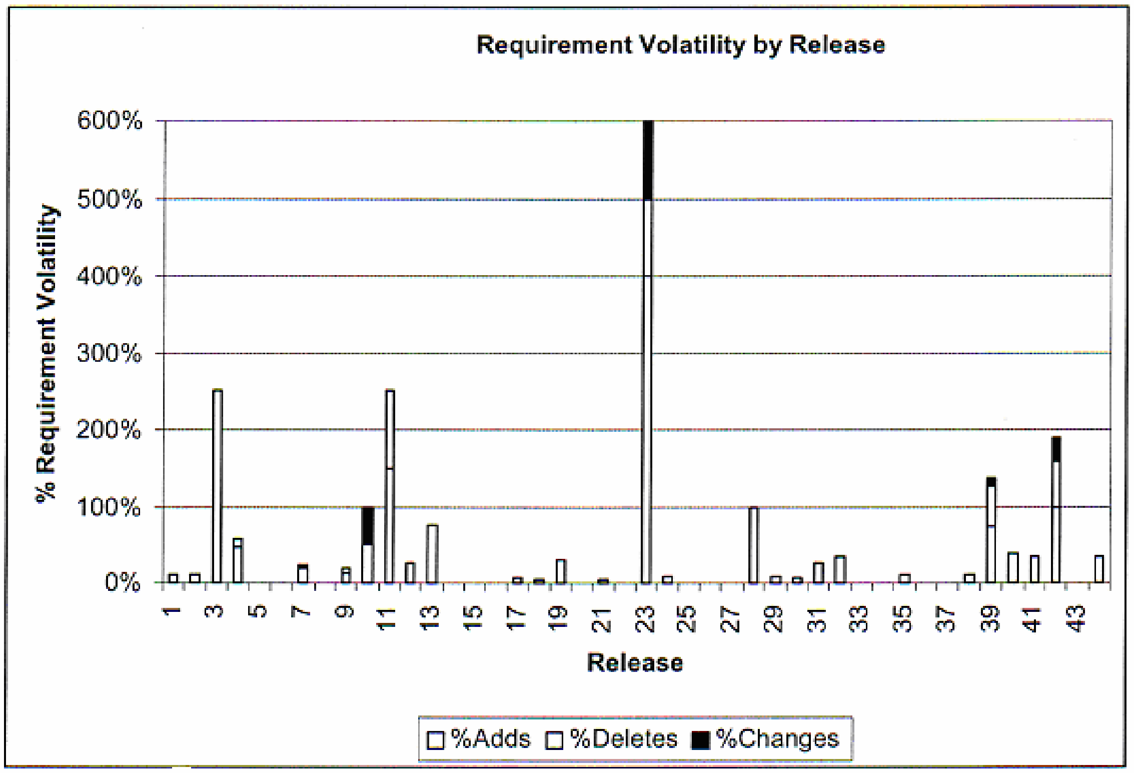


Figure: 3.4. Requirements volatility by Maintenance Release [4]

The average requirements volatility for these 44 releases is 48%. Taken literally, the data from these studies of multiple projects shows that at least one third and potentially as many as half of the delivered requirements implemented during system maintenance were not part of the original plan. The largest observed volatility is an astonishing 600%. Additionally, 250% was observed twice. In the case of release 23, the original plan contained only one new feature that was deleted from the release; four new features were added after design work had begun. During the critical design review, one of the new features was changed from a single user modification to a multi-user function. In combination, the result of these changes was a 78% increase in schedule duration. In the case of release 3, a new high-priority mission was defined during the release and those requirements were added to the version. For release 11, the original requirements were deleted from the release and new requirements were added because of a configuration change in an interfacing system.

Figure 3.5 shows the distribution of these *changes* by category. Additions to the release functionality are the most common form of change, followed by deletions, with scope *changes* occurring least frequently.

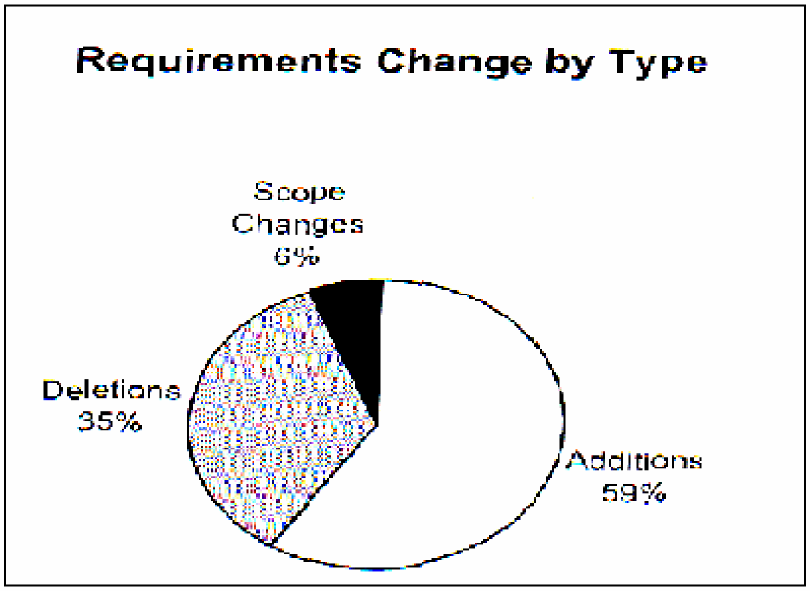


Figure: 3.5. Distribution of Requirements Changes by Type [4]

When in the release cycle do requirements changes occur?

An analysis of data in [4] shows that requirements volatility will affects the time in months of one project that was approved with 17 requirements and planed to be completed on a nine month schedule. Figure 3.6 displays the *changes* in these requirements over the scheduled time for

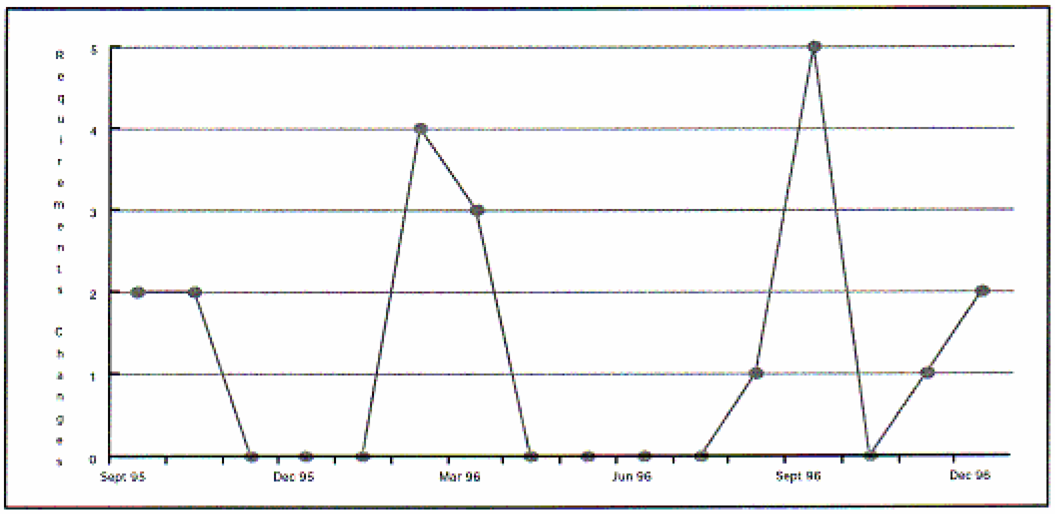


Figure: 3.6.Requirement Changes by Month for One Project [4].

the release. The requirement changes were processed both formally (through the Configuration Control Board) and informally (agreement between users and maintainers).

This chart shows that 20 total changes were made to the release content (volatility of 117%) in the 14 months since project plan approval. This change rate of 1.4 changes/month (about 8% per month) is extremely high.

The two spikes in February and October occurred after design reviews with stakeholders, when major scope changes occurred to some of the requirements. Nine of the changes occurred in the last five months and only six of the delivered requirements were a part of the original approved plan.

3.5 Software Quality

There is no standard definition of quality. Software quality is said to be the customer satisfaction. If a software product meets all the requirements specification and it is bugs free then we can say that the software has high quality. However there is no bugs free software till now. Software requirements volatility and software quality are inversely related to each other. As of requirements volatility increase the reliability of the final product decreases. If the requirements volatility is low, the software quality is high.

When no of bugs is a measure of software quality, it is related mainly to the testing phase. The software is tested for bugs in the testing phase. Testing is performed done even in the coding phase, but that testing is only for that particular program, not directly related to all the system.

Impact of Requirements Volatility	Short Description
Project Schedule	Due to R.V if the schedule of one activity delays, obviously all subsequent activities schedule will be disturbed. Sometimes we need to reschedule the whole project.
Project Performance	Project performance decreases due to R.V. R.V will decrease in later stages of the project. R.V has high impact on coding and maintenance phases.
Project cost	Project cost increases due to R.V.
S/W maintenance	R.V mainly due to Govt. regulations and market competitors. We need to revamp according to new requirements.
S/W quality	Quality decreases due to R.V. We need to re design the test cases according to new requirements.

Table: 3.1. Impact of Requirements volatility

In table 3.1 we show the summary of the impact of requirement volatility on the software project.

Chapter 4

Managing and Controlling Requirements Volatility

Changing the requirements is unavoidable. Therefore we need to find a correct solution to manage changing requirements. However there are number of adequate approaches for dealing with requirements changes in the way that minimizes the impact to the stakeholders. We need to select the correct one to fit for our project.

4.1 Managing volatility in software development models

This section considers several development models to identify their advantages and disadvantages in managing requirements volatility. You can not follow the same procedure (model) for all projects. It depends upon the size of the project, nature of project and more. When you are not following the correct method you have to face more problems in the coding phase. However every model has its advantages and disadvantages.

4.1.1 Waterfall model:

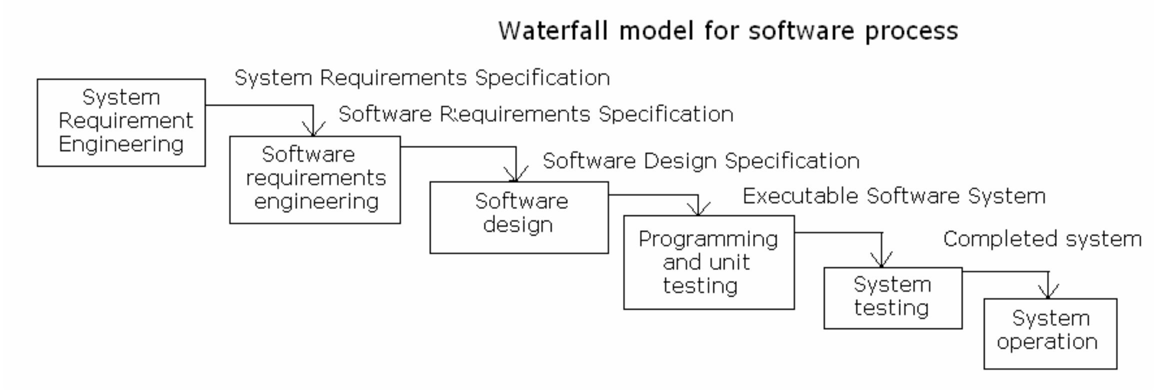

Waterfall model for software process
System Requirement Engineering
System Requirements Specification
Software requirements engineering
Software Requirements Specification
Software design
Software Design Specification
Programming and unit testing
Executable Software System
System testing
Completed system
System operation

Figure: 4.1. Water fall model for software process [20]

The water fall model is designed to follow a set of actives in a sequential order. Starting with system requirement engineering the model ends with system operation. This model clearly explains what all the activities are in each phase.

The waterfall model is an attempt to put discipline into software development process by forcing standard documentation. Before you are going to the coding phase, the design documentation has to be written. Each module has to be tested before going to the next sequent module. The typical programmer would prefer the coding before the documenting. Consequently some programmers consider the whole model to be painful, because it tries to force a discipline.

The waterfall create problems because the requirements are often learned after the implementation phase (in fact requirements are never completed), when there is a significant repeated backtracking, when new requirements are continually discovered.

My suggestion is to not follow the waterfall model strictly when the requirements are not well defined and understood, otherwise the requirement volatility impact will be high. How ever the waterfall model is suitable for small projects like student academic projects because the requirements not change much.

4.1.2 Built-and-fix model: This model is helpful when the program is small enough to be written entirely by one person. The basic cycle is:

1. Built the first version of the software

2. Modify the software as new requirements occurs during the development

3. Repeat the second step till no requirements volatility is found or till the customer is satisfied.

This model is simple, the important thing is that you need to modify the software many times (the number of releases increases). You also do not need to wait long time to implement new requirements.

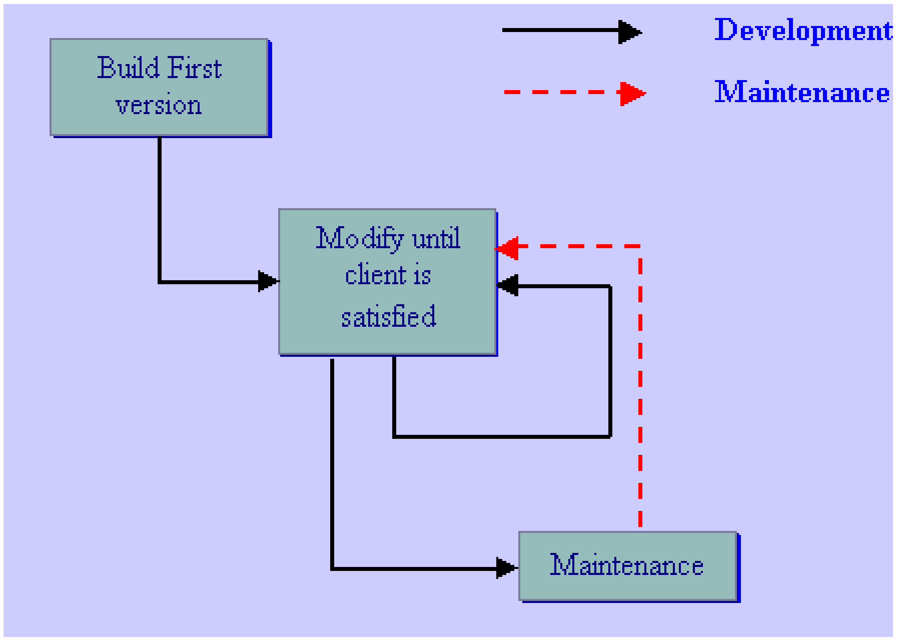


Figure: 4.2. Built-and-Fix Model [26].

This model is more suitable than the waterfall model for projects with new requirements and changing requirements because the process is iterative and incremental.

4.1.3 Spiral model:

The spiral model is useful for large and long term applications. The spiral model compares the features of prototyping model and water fall model.

1. The preliminary design is created for the new system according to preliminary requirements.
2. The first prototype of the system is constructed from the preliminary design.
3. The second prototype is evolved by a fourfold procedure: (1) evaluating the first prototype in terms of its strengths, weaknesses, and risks; (2) defining the requirements of the second prototype; (3) planning and designing the second prototype; (4) constructing and testing the second prototype.
4. The same procedure is repeated until the customer is satisfied with the product [22].

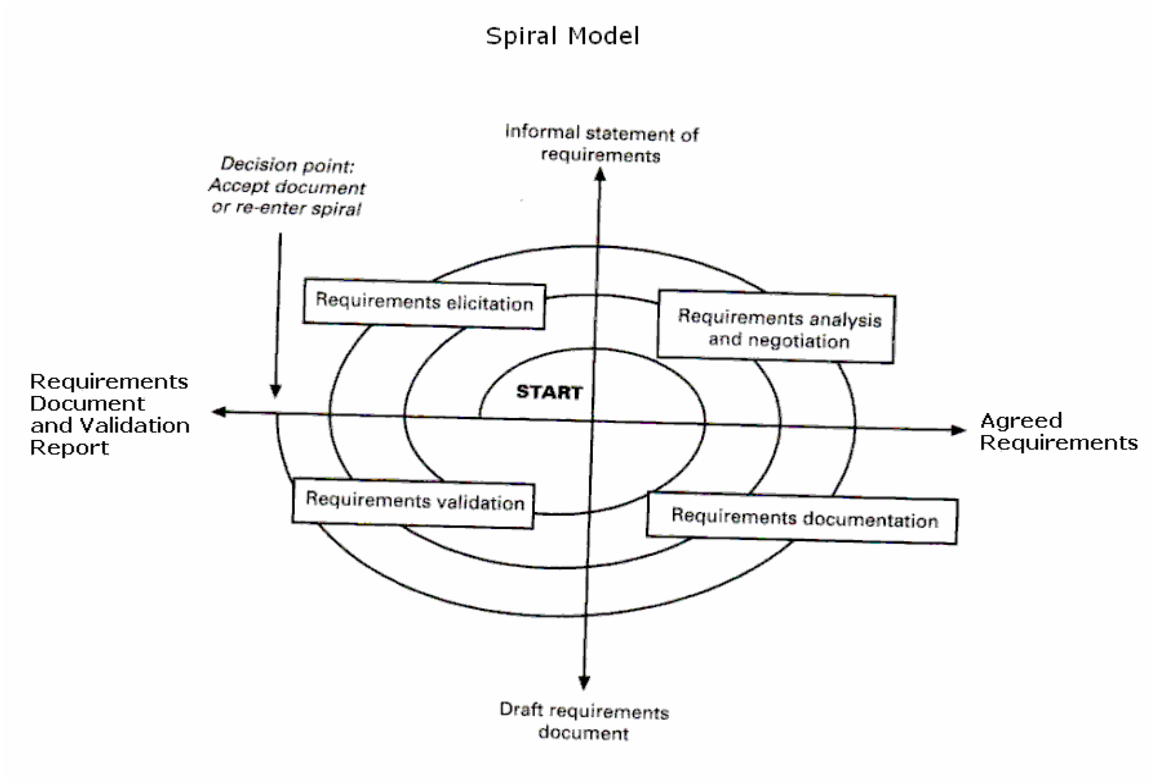

Spiral Model
Informal statement of requirements
Decision point: Accept document or re-enter spiral
Requirements elicitation
Requirements analysis and negotiation
START
Requirements Document and Validation Report
Agreed Requirements
Requirements validation
Requirements documentation
Draft requirements document

Figure: 4.3. Spiral model [20].

Concluding, also this model is suitable to changing requirements because it is an iterative process. If new requirements arrive late, it is enough to add another round of the spiral to implement the new requirements.

4.1.4 Extreme programming

Extreme programming is a disciplined approach to software development. Extreme programming is perfect when the projects have dynamic requirements and is successful because it has simple rules and practices. This process emphasizes customer involvement and promotes team work. With extreme programming we can decrease the unproductive activities; therefore we can reduce the cost of the project [21].

Extreme Programming (XP) argues that preplanning which is the cornerstone of each the various disciplined developing methods (such as the Waterfall model), is a waste of time. Most likely, the results of planning will be thrown out as new requirements are discovered. XP consists in simply building software on requirements that are understood at the time that programming commences.

Unfortunately, writing code is not easy before it is known what the code is supposed to do. Thus, the simplest architecture, sufficient to deal with all the known requirements, is used without too much consideration of possible changes in the future and making the architecture flexible enough to handle these changes. Too much consideration of the future is a waste of time, because the future requirements may never materialize and an architecture based on these future requirements may be wrong.

What happens when a requirement comes along that does not fit in the existing architecture? XP says that the software should be refactored. Refectory consists in stepping back, considering the new current full set of requirements and finding a new simplest architecture that fits the entire set.

The code that has been written should be restructured to fit the new architecture, as if it were written with the new architecture from scratch. Doing so may require throwing code out and writing new code to do things that were already implemented. XP’s rules say that refactoring should be done often. Because the code’s structure is continually restructured to match its current needs, one avoids having to insert code that does not match the architecture. One avoids having to search widely for the code that affects and is affected by the changes. Thus, at all times, one is using a well-structured modularization that hides information well and that is suited to be modified without damaging the architecture.

However, refactoring itself is painful. It means stopping the development process long enough to consider the full set of requirements and to design a new architecture. Furthermore, it may mean throwing out perfectly good code whose only fault is that it no longer matches the architecture, something that is very painful to the authors of the code that is changed. Consequently, in the rush to get the next release out on time or early, refactoring is postponed and postponed, frequently to the point that it gets harder and harder. Also, the programmers realize that the new architecture obtained in any refactoring can be wrong for some future new requirements [24].

4.2 Managing volatility in software development Phases

4.2.1 Analysis and design Phase

Requirements analysis begins after the requirements collection. During the requirements analysis we can identify the risks of the project. The purpose of the analysis phase is to ensure that requirements are complete, consistent and the problem is understood by the development team. In the analysis phase the developers will decide the duration of project, staff hours, estimation of system size.

Design is the creative process of transforming the problem in to a solution; the description of the solution is called design [18]. In this phase we prepare preliminary design review, and select architecture to follow.

A survey by Zowghi, and Nurmuliani, [11] said that requirements changes during the analysis phase are high.

Architecture and design of software product fully depends upon the Requirements specification. When there is high requirements volatility we need to change the design continuously. This is very difficult to handle without correct planning. Requirements are learned after the implementation phase, but there is requirements volatility in maintenance phase as well.

Beyer [23] says that rather than waiting for the complete requirements set to do the design we should start the design with the collected requirements, start coding, and later modify the design according to the new requirements. In this way we save the time for not waiting for the full set of requirements. At same time you can handle requirements changes whatever requirements you collected.

When we are developing a software product similar to one which already exists, we can use the technique called “prediction by analogy“. We can study and investigate the final requirements specification documents and product which already exist. With this method we can decrease some volatility of requirements. If we analyze several historical products we can decrease the chances of requirements volatility.

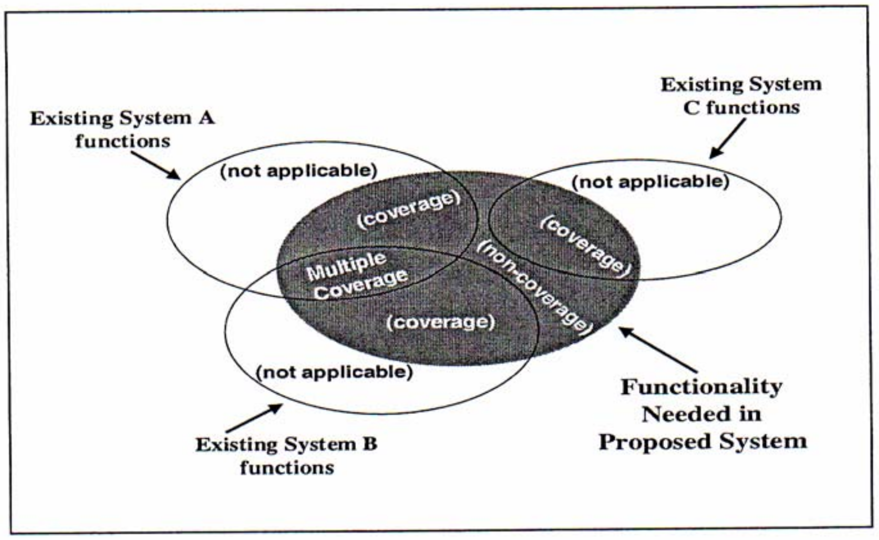

Existing System A functions
Existing System C functions
(not applicable)
(coverage)
(not applicable)
(coverage)
Multiple Coverage
(non-coverage)
(coverage)
(not applicable)
Functionality Needed in Proposed System
Existing System B functions

Figure: 4.4. Prediction by analogy [9].

4.2.2 Coding phase

The programming part will start in the coding phase. The coding phase is the critical phase in the software development. Each requirement change in requirements specifications will effect the coding.

As a program undergoes continual change, its structure decays to the point that is very hard to add something new or changing something existing without effecting seemingly unrelated parts of the program in a way that causes errors. It can be difficult even to find all places that need to be modified. If the programmers make poor guesses (for example, declaring data as natural number rather than integer when we need integer) the program behaves in strange and unpredicted ways.

These days the programming languages have been developed to make easy to program for the developer. The object oriented programming languages, for instance supports the reusability of code. For example when you need to develop a pay slip for junior officer, you do not need to spend time to develop; you can reuse already existing code from other employees pay slip. SAP (System Application product) which is already built in product, we need to modify that according to the customer requirements.

Structured programming: Structured programming will help you to write the code according to the new requirements generated during the software development, I.e. developing the new code or modifying the old code. Here there are two ways to develop the system according to the new requirements with benefits and defects.

1. Patch the change into the code in the traditional way after a careful analysis of ripple effects; observe that the documented structured development assists in finding the portions of the code affected by and affecting the change.

2. Redo the entire structured development from the top downward; taking into account the new requirement and the old requirements, all in the right places in the refinement.

The problems with the first choice are that:

1. No matter how careful we are, some ripple effects are always overlooked.

2. Patching destroys the relation between the structured development and the code, so that the former is no longer accurate documentation of the abstractions leading to the code. Thus, the new code contradicts the nice structure imposed by the structured development. Moreover, the names of the high level abstractions that get refined into code no longer imply their refinements.

Therefore, the correct alternative was the second, to start from scratch on a new structured development that recognizes the modified full set of requirements. However, then the likelihood is that the new code does not match the old code. Most structured programmers do this redevelopment with an eye to reusing as much as possible. That is, whenever one encounters a high-level statement with identical name and semantics as before, what it derived before is put there. However, the programmer must be careful to use the same high-level statements as refinements whenever possible. Even though the second alternative is good, it’s turned out to be so painful to take care of all these things.

In the first choice we need to do the modification by patching up the code, and then go back and modify the original structured development to make it look like the new program was derived by Structured Programming. We are also required to do extra work for external reasons e.g., preparing a paper for publication or a contract.

4.2.3 Testing phase

In testing phase we test the whole system for technical, functional, and others. The quality of product can be decided in this phase. If the projects meet all requirements specification and with bugs free then we can say software has good quality. Still now there is no 100% bugs free software.

For the testing phase test cases are written at the designing stage. If there are any changes in the requirements automatically there should be changes in the coding and design phases, so we need to modify the test cases again.

If the requirements changes occur after the testing phase, the test cases are not correct according to the new requirements. So we did not test our system correctly. We can not say that our system is 100% bugs free and has good quality.

4.2.4 Maintenance phase

Most large software systems have long life times during which the software undergoes significant changes in the form of maintenance. Software maintenance consists in: including the changes; adding/deleting the functionality to the software, correcting the defects discovered in the software system, modifying the existing functionalities, adapting the software to changes in the environment and changing the software to support future maintenance operation.

In maintenance phase, requirements collection is very difficult because there are continuous changes to requirements. In maintenance environment change requests are generated from various people including decision–makers, system operators, maintainers, customers, and business partners. Here the requirements collection is difficult because the people are from different backgrounds and the level of understanding associated with computers is different.

In the maintenance environment, requirements are gathered through Change Request Forms (CRF) (see fig: 5.5) that identify the future additions/deletions and/or modifications and defect corrections. As we said earlier Change requests are gathered from a variety of people. These people have different backgrounds and levels of understanding associated with computers and system operations. This diversity often leads to misinterpretation of the intent and scope of the change request by the change request team resulting in incomplete and incorrect requirements. This misinterpretation impacts the size and effort associated with implementing the requirements.

Project Name:	**Project No.:**	**Project Manager Name:**
Requestor Name:	**Request Date:**	**Resolution Requested By:**
Description of Change:		
Reason for Change:		
Impact on Scope and/or Deliverables:		
Impact on Resources and Quality:		
Impact on Resources and Quality:		
Disposition of Change Resolution:	**Accepted:**	**Denied:**
Signature of Project Manager:	**Date:**	
Signature of Project Sponsor:	**Date:**	

Figure: 4.5. Change Request Form [25].

Maintenance of a new system: The maintenance of new system follows immediately the implementation stage. In this type of maintenance we do not need to do rigorous work for maintenance process. Because, in most cases the people already had the results of previous SDLC (Software Development Life Cycle) phase in hand and the people who worked in development phase will work also in maintenance phase. We need to do only small changes (addition, deletion, and modification) to the existing product.

Maintenance of an old system: Some software product exists from long time; probably there will be many changes in functional, government regulations, market competitors, almost in all aspects. We need to do a well structured work for good maintenance results (modifications).

Modifying items typically requires changes to the existing project performance Specification (PPS) and Interface Design Specification (IDS) or both. System engineers analyze the impact and interpret the precise meaning of each upgrade item during the system specification and software specification phases. The analysis is documented in the PPS and IDS, Which are reviewed at the preliminary design review (PDR). The PPS and IDS are the basis for implementing a new baseline.

Detail design of upgrade items takes place following the PDR. Detail design of the functionality documented in the PPS and IDS typically require changes to the PPS and IDS. The PPS and IDS pages requiring changes and detailed design are reviewed at the Critical Design Review (CDR). Successful completion of the CDR, as judged by the customer representatives and management, initiates the implementation phase.

The new versions of the PPS and IDS presented at the PDR and the CDR are changed at the PDR, the CDR and Post CDR. These Specification changes are documented using the specification change form and implemented using well defined specification changes [3].

Process model/Phases of the process	Suggestion	Reference
Waterfall model	It is not suitable to handle volatile requirements. Suitable for well defined projects.	[20]
Built-Fix Model	This is suitable for very small projects. Modify the software until the customer is satisfied.	[26]
Spiral model	It is good to handle R.V. First develop the first prototype with preliminary requirements. Collect the new requirements, develop the second prototype with new requirements including the missing requirements or errors requirements in first prototype. This is well suitable for big projects.	[22]
Extreme programming	First start to develop the project according to preliminary requirements. Try to make design and built the architecture according to it. Even though coding with out knowing exactly what to develop is difficult. It is suitable to manage volatile requirements.	[24]
Analysis and design phase	Start design with preliminary requirements and modify the design according to new requirements. Divide the system into modules.	[11]
Coding phase	Reusability of code, modify the old code according to new requirements or redevelop the whole code.	—
Maintenance phase	Update the previous PPS and IDS.	[3]

Table: 4.1. Managing and controlling requirements volatility.

Chapter 5

Summary and Conclusions

In this thesis I have described several aspects of requirements volatility in particular the causes and impact of requirement volatility on software development process. I have also suggested several ways to handle requirements volatility.

The causes of requirements volatility can not be overcome fully but we can minimize some causes like technical aspects and poor communication between stake holders and developers (most of these are internal factors as specified in 2.2).

Requirements volatility has impact on the whole software development life cycle not excluding even one phase. It mainly affects the coding and maintenance phases of large and long term projects. Large projects need to be divided in to modules to decrease the effects of requirements volatility. In this way, the impact will be reduced to one module.

Requirements volatility has impact on project schedule, project cost, and quality. It can not be avoided, but we can minimize the effect of requirements volatility by following some methods in analysis, design, and coding. Due to the impact of requirements volatility many projects have failed.

Among the methods suggested to manage requirements volatility, the waterfall model has the drawback that forces to follow the documentation in certain order. Also XP, and spiral model have some drawbacks but they are more suitable in managing volatility.

More research is needed to develop the flexible architecture which is suitable for requirements volatility. We need to define new methods to manage requirements volatility.

References

[1]. **T.Javed Manzil, M. Quiser, and S. Durrani,** "A study to investigate the Impact of requirements Instability on Software Defects", ACM SIGSOFT Software Engineering Notes, 29(3), May 2004, pp:1-7.

[2]. **Lamswede, A**. Requirements Engineering in the Year 00: A research perspective. In proceeding of the 22nd International conference on Software Engineering (ICSE'2000), Limerick, Ireland, 5-19, ACM Press.

[3]. **J. Henry and S. Henry,** "Quantitative Assessment of Software Maintenance and Requirements Volatility", Proceedings of the 1993 ACM conference of Computer science, Indiapolis, Indiana, United States, 1993 pp: 346-351.

[4]. **G. Stark, P. Oman, A. Skillicorn, and R. Ameele,** "An Examination of the Effects of Requirements Changes on Software Maintenance Releases" in Journal of Software Maintenance Research and Practice, Vol. 11, 1999, pp:293-309.

[5]. **R.J. Costello, and D. Liu,** "Metrics for requirements engineering", in Journal of Systems and Software, 29(1), April 1995 pp: 39-63.

[6]. **Nidumolu, S**."Standardization, Requirements uncertainty and software project Performance", Information and Management, vol.31, pp.135-150, 1996.

[7]. **S. Ferreira, J. Collofello, D. Shunk, G. Mackulak, and P. Wolfe.** "Utilization of Process Modeling and Simulation in Understanding the Effects of requirements volatility in Software Development", International Workshop on software process Simulation and Modeling (proSim 2003), Portland, Oregon,USA, May 3-4 2003.

[8]. **Karen Reid**. 2005. " lecture slides by Dr.Yiujun Yu Yun in University of Toronto for Fall 200472005 on Software Engineering. Created in May 15, 2005. Viewed on April may 20 2005.

http://www.cdf.utoronto.ca/~csc408h/winter/lectures/Week4-management.pdf.

[9]. **Frank Armour, Bill Catherwood and Carig Beyers**. "A Frame work Managing Requirements volatility using function point as currency".

[10]. The standish group. The CHAOS report. Dennis, 1994.
http://www1.standishgroup.com/sample_research/PDFpages/chaos1994.pdf

[11]. **Ian Sommerville, Lutz Prechelt**. "Software cost estimation", 2004.

[12]. **Linsy Iansmith**. "A tutorial on Principal Competent Analysis by ", Feb 26, 2002.

[13]. **F. Lanubile and G.Visaggio. "**Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned" System Software, 1997 38: pp: 225-234.

[14]. **Agresti, A.** Categorical Data Analysis, John Willy &sons, New York,1990.

[15]. **R.Creel.** Requirements Volatility. Jan 16 2004. Viewed Feb. 2005
http://research.it.uts.edu.au/re/re-archive/1656.html,.

[16]. **V. Ambriola, V.Gervasi**. “Process Metrics for Requirement Analysis” 7th European workshop on Software Process Technology, Kaprun, Austria, Feb 21-25 2000, pp: 90-95.

[17]. **Y.K. Malayia and J. Denton**. “Requirements Volatility and Defect Density”10th IEEE International Symposium on software reliability Engineering, Boca Raton, Florida, Nov 01-04, 1999, pp: 28-39.

[18]. **Pfleeger, S.L** . Software Engineering theory and practice, prentice Hall, 1998.

[19]. **D. Powell.** Software Process Management, June 1, 2004. Viewed on March 2005. http://www.int.gu.edu.au/courses/2203int/req.

[20]. **Gerald Kontonya and Ian Sommerville**. Requirements Engineering Process and Techniques .Wiley Publications Reprinted February and October 2002.

[21]. **Kent Beck**. Extreme programming: A gentle Introduction. Feb 28 2004. http://www.extremeprogramming.org/index2.html.

[22].**seachVB.com**.2005.http://searchvb.techtarget.com/sDefinition/0,sid8_gci755347,00.html

[23]. **D. Zowghi, N. Nurmuliani**, “A study of the Impact of requirements volatility on Software Project Performance”, Proceedings of the Ninth Asia-Pacific Software Engineering Conference , APSEC 2002, Gold Cost, Queensland, Australia,04-06 Dec 2002, pp:3-11.

[24]. **Beck, K** .Extreme programming Explained: Embrace Change, Addition- Wesley, Reading, MA (1999).
[25]. **gantthead.com**.2005.
http://www.gantthead.com/templates/download.cfm?ID=205416.

[26]. **Adrin Als & Charles Greendige**. Built and Fix Model, 2003. Viewed on April 2005. http://scitec.uwichill.edu.bb/cmp/online/cs22l/BuindAndFix.htm. Last modified: Feb 2005.

[27]. **Loconsole. A and Börstler. J**. “An Industrial case Study on Requirements Volatility Measures” Umeå University, Internal Report, Feb 2005.