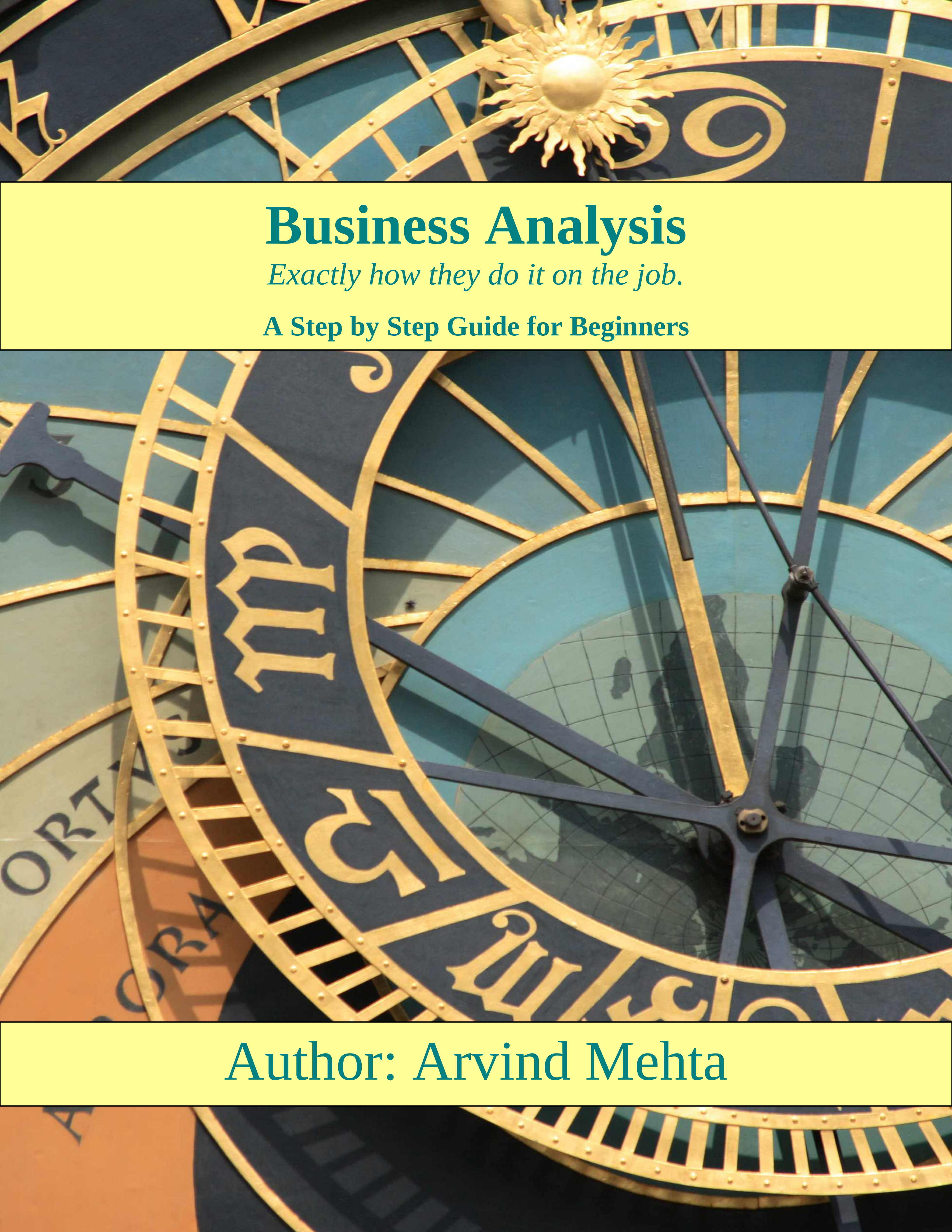
Business Analysis

Exactly how they do it on the job.

A Step by Step Guide for Beginners



Author: Arvind Mehta

Business Analysis

- Exactly how they do it on the job.

***A Step by Step Guide
For Beginners***

*Includes Business Analysis Techniques,
Process Modeling with UML Diagrams,
Software Testing and Project Templates*

Author:

Arvind Mehta

Business Analysis- Exactly how they do it on the job.

First Edition

Copyright © 2013

ISBN-10: 0-615-80726-7

ISBN-13: 978-0-615-80726-3

The right of Arvind Mehta to be identified as author of this work has been asserted by him in accordance with the U.S. Copyright Act of 1976.

All rights reserved. No part of this publication may be reproduced, stored in retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording including but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning or otherwise, without the prior written permission of the author. You may use the work for your own noncommercial and personal use; any other use of the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

All trademarks used in the book are the properties of their respective owners. The use of any trademark in this text does not vest in the author any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

This book is written solely for informative purpose. Author of the book is not engaged in rendering any business services thus is not responsible for any sort of tangible or intangible damage caused by the information provided in this book.

Dedicated to my lovely family-

My wife Vandana and my daughters Missy, Molly and Gunjan

About the Author:

A graduate in Business Management from Institute of Productivity and Management (IPM), India, Arvind Mehta has more than twelve years of experience in business process modeling & designing process architecture, process re-engineering & optimization and aligning strategic IT initiatives with organizational objectives specifically with supply chain processes in SAP® centric environment. Arvind is also a certified scrum master (CSM) and is a participating SAP® Supply Chain Management consultant with avid interest in continuous process improvement.

Arvind is also the founder and chair of the board of Association of Supply Chain Management Practitioners (ASCMP), Omaha, Nebraska based organization engaged in generating and encouraging a highly creative and analytic thought leadership, promoting the best practices in supply chain arena. ASCMP provides a comprehensive knowledge sharing platform to the professionals and organizations to collaborate and work together for mutual benefits. It also educates small and mid size companies about supply chain best practices.

Preface:

The prime motive which gave birth to the idea of writing this book came from my passion and interest in continuous process improvement. While assisting my clients to maximize their return on investment on their strategic IT initiatives focused on their business processes optimization programs, I decided to document my experience which finally led to authoring this book.

Purpose of the book:

Writing this book is an endeavor to reaching the group of people who wish to start their career in IT as a Business Analyst as well as those who are already working as business analysts and intend to have a more clear understanding of business analysis.

This book is a compilation of my professional experience and notes which I gathered during my work. It aims at providing a step by step practical understanding of what a Business Analyst does on his job in his day to day routine. The explanations in this book have been supported with the help of real project documents and templates and business process examples. I have put in my best efforts to use an easy going language and tried my best to compress the hundred points in one page instead of spreading one point in hundred pages.

Scope of the book:

There are seven chapters in this book. **Chapter one and two** explain the basic concept of IT projects, various key roles involved in a typical project, and pre project activities to be performed by a business analyst. **Chapter three and four** focus on the requirement gathering process, techniques, managing the requirement risks, scope, documenting the requirements and cover gap analysis & use cases. **Chapter five** explains the software development life cycle, various software development models and also suggests the pros and cons of each model. **Chapter six** explicates business process modeling through UML diagrams and **Chapter seven** covers manual software testing, software defect life cycle, test plan, test strategy, test cases and test conditions.

I believe this book should serve the purpose that planted the seed of the idea of writing this book and you will find it as one of the most easy going and interesting books with the most practical aspects of business analyst profession.

Acknowledgement:

I extend my heartiest thanks to my friend SN Padhi who encouraged me to write this book. Surya is SAP® FICO consultant and has been dedicated to serve the SAP community by writing books and developing free online tutorials and contents for SAP® professionals. Some of the contents in this book is a result of my research on internet and has been derived from the various internet sites and white papers and may or may not exist in the exact same language as I found their existing lingo to be the best expression of the concept and can not be expressed in a better way than their existing form. I express my heartfelt gratitude to the authors of those articles and white papers for authoring the contents as those were of tremendous help to me in writing this book.

I also thank to all others who were involved directly or indirectly and extended their support to writing this book, specially my family- my wife Vandana and my daughters Missy, Molly and Gunjan. I couldn’t have been able to dedicate my time to writing this book without their understanding and cooperation.

Contents at a Glimpse

Page

Chapter 1: Introduction **13**

Chapter 2: Pre-project activities **19**

Chapter 3: Requirement Gathering **33**

Chapter 4: Documenting Requirements **65**

Chapter 5: Software development Life Cycle and methodology **77**

Chapter 6: UML Diagrams For Business Process Modeling: 101

Chapter 7: Software Testing: 113

Annexure: Sample Project Documents and Documents:

CHAPTER 1:

Introduction

1.1 IT Project: Understanding the bigger picture

To understand what the role and responsibilities of a business analyst are and where a business analyst fits in a project, first we need to look at the project as a bigger picture and try to understand:

- What a project is
- Why a project is initiated
- What other key roles a project involves
- How these roles are aligned with each other

Defining Project:

Project management book of knowledge defines a project as "A temporary endeavor undertaken to create unique product or service- temporary means it has an end date and unique means the result of every project is different than any other project or other functions of the organization"
As we progress through the project life cycle, our understanding of the project, its characteristics and its result keeps getting elaborated gradually. Projects are different than operations- projects are temporary in nature and usually involve heterogeneous teams where as operations are ongoing activities and usually involve homogeneous teams.

Why a Project is initiated:

Initiating a project carries a unique problem solving approach and is usually aimed at resolving a business related issues which can be of any type such as:

- Developing a software application to automate the sales process and integrating it with inventory management application so that right at the time of taking the order from the customer, sales person would know the exact date of delivery.
- Integrating the functionalities of different departments to make a seamless business process so that management can get the real time reports which in turn will speed up the decision making process.
- Recognizing the need of periodic training of employees to increase their efficiency and productivity.
- Analyzing the current business processes and sequence of activities to find out the scope of any further improvement/ optimization possible in order to minimize the total cost of ownership and maximizing the return on investment.

1.2 Key Roles Involved:

Typical key roles may involve but may not be limited to the described below:

Project Sponsor/Executive Sponsor/Project Champion/Owner:

The project sponsor reports to the corporate executive or executive committee. Executive sponsor may or may not be the project sponsor but usually is always a project champion or owner. On a very large project an executive sponsor may delegate the tactical details to a project sponsor with defined responsibilities. He has overall responsibility for the project at the management level including funding, go/no go decision making and providing resource support.

Project Manager:

The project manager is responsible for planning, executing, managing and closing the project. He manages day-to-day activities of the project ensuring that requirement related tasks are delivered on time, within budget, and within scope and as per the required standards. The project manager must ensure that proper stakeholder approval of the requirements is obtained before progressing forward with project delivery.

Business Analyst:

The business analyst functions as a bridging element between the business users and technical team. Business analyst elicits, analyzes, documents and reviews the requirements for accuracy and presents them to project stakeholders for review and approval.

Programmer:

Programmers are the technical resources assigned to a project, and may include many technical roles within a project team, e.g. The Technical Lead oversees the design, code, and test activities for the technical members of the project team. Developers also plan the application's transition to the user community, often working directly with business analysts and trainers.

Quality Assurance Analyst:

The quality assurance analyst is responsible for ensuring that quality standards are adhered to by the project team.

Trainer:

The trainer is responsible for developing user training curriculum materials and delivering training to end user personnel. These materials are based on the functional requirements.

Technical Architect/ Application Architect:

The application architect defines the architectural approach and high-level design for a project solution. The application architect is responsible for determining the technical direction of the project and the overall structure of the solution.

System Administrator:

The system administrator grants the rights and permissions in order to provide different security levels to different roles to ensure secure access of the application.

Database Analyst (DBA):

The database analyst is responsible for all technical aspects related to designing, creating and maintaining project databases.

Infrastructure Analyst:

The infrastructure analyst designs the overall hardware and software infrastructure and environment needed to meet the application development and operational requirements.

Data Modeler/ Information Architect:

The information architect is responsible for assessing the overall data requirements of an information system project. Information architect identifies reusable data assets and resolves enterprise data modeling issues.

Solution Owner:

The solution owner is responsible for defining and approving the project scope and ensuring that it aligns with the business strategy. Approving project scope changes and defining the project success criteria and measurement are also part of the responsibility of the solution owner. (also see Executive Sponsor).

End User:

The end user represents the group of people in the organization (and often external to it also) who will actually interact directly with the software application.

Subject Matter Expert (SME):

The subject matter expert (SME) provides expertise in a particular business functional area. SME responsibilities are closely tied to defining, approving and using the functional requirements for the project. SMEs typically work very closely with business analysts in identifying and managing the requirements.

Stakeholders:

Stakeholders represent anyone materially affected by the outcome of the project. Stakeholders are often a prime source of information when planning and managing requirements.

1.3 Where a Business Analyst fits in the Bigger Picture:

Now let's see where the business analyst fits in this bigger picture. When a company or a department recognizes the need of initiating a project, the high level activities take place in the following sequence:

1. Project sponsor prepares the project proposal and gets it approved by executive project sponsor
2. Once the project proposal is approved, a steering committee is formed which writes project charter
3. A project manager is appointed and project charter is handed over to the project manager
4. Project kick off meeting is conducted
5. Project scope is defined
6. High level schedule is developed
7. Quality standards are identified
8. Project budget is established
9. Possible initial risks are identified
10. Project manager develops the project plan
11. Stake holders involvement is identified and documented
12. Communication plan is developed
13. Business case is refined and reviewed
14. Project goes to the project sponsor for approval
15. Once the project is approved, project team is formed
16. Business analyst joins and receives the scope statement from project manager
17. Business analyst identifies the stake holders and SMEs to understand company's current business process (AS-IS Report)
18. Business analyst gathers the requirement about future business processes and new/changed characteristics of the software application (TO-BE Report)
19. Business analyst performs Business Process Modeling (BPM) to suggest the process improvements and optimize business processes
20. Business analyst performs the GAP analysis
21. Business analyst writes business requirement document/ functional requirement specification
22. Business analyst develops UML diagrams to communicate requirement to the technical team
23. Application architect designs the application
24. Application is divided into different modules and these modules are allocated to the programmers to be developed on the development server
25. Programmers develop the modules and perform unit testing

26. Once the modules are unit tested, they are allocated to testers for other type of testing on test servers
27. Once the application is tested and all the modules are integrated, application is handed over to the business users for user acceptance testing (UAT) with actual business data
28. Once the application is ready to use, project goes live on the production server
29. Business users are trained to work on the new application
30. Application support team is assigned the task of application maintenance

If you look at the point no. 16 through point no. 22, you can have some idea about where the business analyst fits in the bigger picture and what his role and responsibilities are.

Now from chapter 2 onwards, we will cover the responsibilities of a business analyst step by step in more details.

CHAPTER 2:

Pre-project Activities

2.1 Analyzing the Enterprise

It is crucial for a business analyst to understand the organizational environment, how the project, different key roles involved in the project and their work in the project supports the entire organization. Enterprise analysis is a pre-project activity involving business architecture initiative which helps to capture the snapshot of the business to provide platform to create meaningful business and functional requirements.

During enterprise analysis, business requirements for future project investments are recognized and documented in alignment with business goals, objectives and needs for that investment in order to identify the most valuable return yielding projects. Enterprise analysis increases the value; projects bring to the organizations in order to develop innovative business solutions to satisfy the never-ending demand for efficient products and services.

While conducting Enterprise analysis, the focus is at the enterprise level where considerations about proposed initiatives traverse across the enterprise. Business analyst plays an important role working with stake holders and subject matter experts to provide management with the decision-support information such as business implications, inter-project dependencies and system interfaces gathered from the internal organizational resources and industry expert's suggestions.

2.1.1 Defining Business Architecture

Business architecture is the responsibility of business architects who are usually senior business analysts having the knowledge of:

- General Business Practices
- Industry Domains
- IT-enabled business solutions
- Current and emerging business concepts, strategies and practices
- How various lines of business within the organization interact with each other
- Business concepts for organizing enterprise knowledge
- Standard architectural principles and semantics, including an understanding of how business issues derive information system (IS) requirements
- Standard business concepts and guidance as to how to use them to create organized information about specific enterprise

Definition:

Enterprise business architecture is a set of documentation that defines:

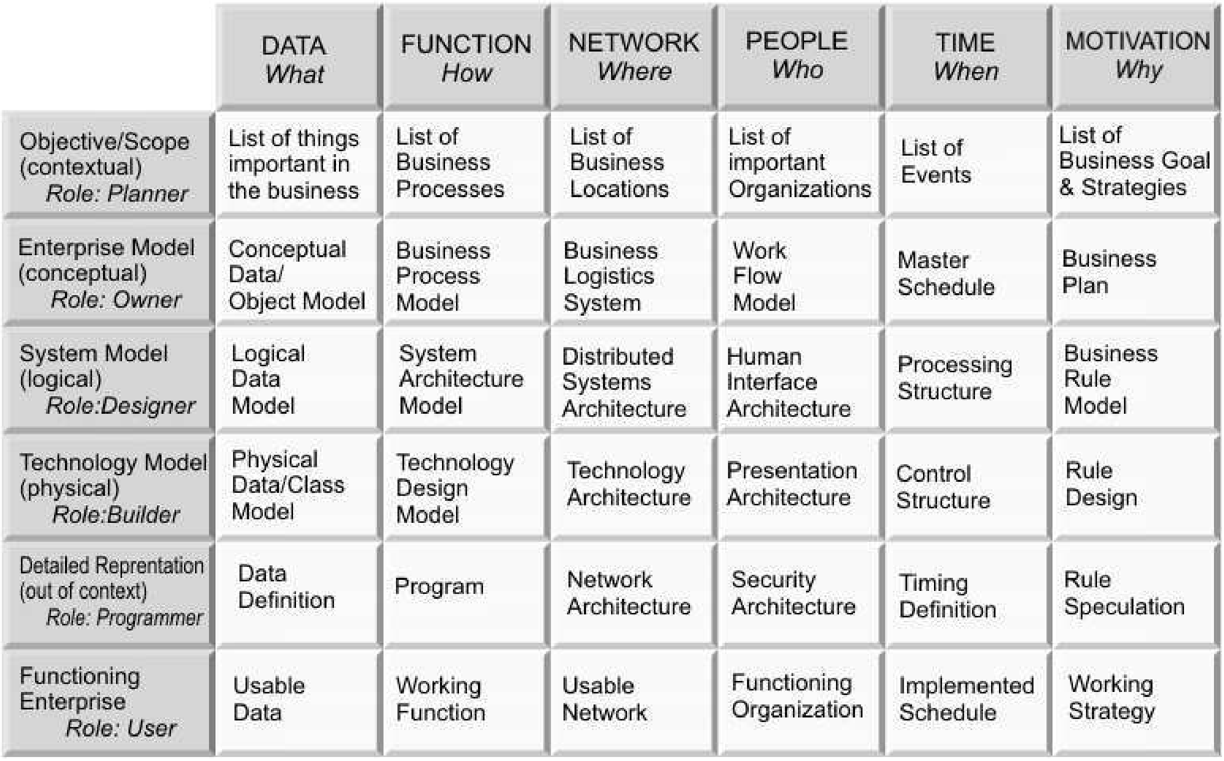
- Vision and Mission
- High level functional view of organization
- Organization's current and future capabilities
- Long term goals and objectives
- Rules, policies, procedures and processes
- Business strategies
- Technological environment
- External environment
- Competencies
- Stake holders

Purpose:

The purpose of business architecture is to provide unified structure and contexts that guide selection and management of programs and projects.

Techniques:

1. **Zachman Framework:**



	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*
Objective/Scope (contextual) *Role: Planner*	List of things important in the business	List of Business Processes	List of Business Locations	List of important Organizations	List of Events	List of Business Goal & Strategies
Enterprise Model (conceptual) *Role: Owner*	Conceptual Data/ Object Model	Business Process Model	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
System Model (logical) *Role:Designer*	Logical Data Model	System Architecture Model	Distributed Systems Architecture	Human Interface Architecture	Processing Structure	Business Rule Model
Technology Model (physical) *Role:Builder*	Physical Data/Class Model	Technology Design Model	Technology Architecture	Presentation Architecture	Control Structure	Rule Design
Detailed Reprentation (out of context) *Role: Programmer*	Data Definition	Program	Network Architecture	Security Architecture	Timing Definition	Rule Speculation
Functioning Enterprise *Role: User*	Usable Data	Working Function	Usable Network	Functioning Organization	Implemented Schedule	Working Strategy

Zachman framework provides common structure and classification scheme for descriptive representation of an enterprise. This framework establishes a relationship between several components of an organizational structure. Zachman framework helps to understand how a fundamental design of an enterprise leads to a well integrated and well functioning organization by unifying these components in one framework.

2. POLDAT Framework:

An alternative, simpler structure that is often used in business process reengineering projects is the POLDAT (Process, Organization, Location, Data, Application, Technology) framework for business process reengineering, referred to as the ***Hexagon of change*** or ***Six domains of change.*** This framework provides us with a approach to defining the scope of business transformation initiatives. The framework yields artifacts such as documents, tables, matrices, graphs, models and organizes them into the following domains:

Process:

The business processes that flow value from the organization to the customer. What are our current processes and do we have opportunities to improve those processes? What is the desired state of business operations?

Organization:

The organizational entities that operate the business processes including the management teams, staff positions, roles, competencies, knowledge and skills. What changes in culture, competencies, capabilities, teams, organizations and trainings are required to achieve the essential business changes? What support systems are required to accept the new business solutions and ensure they operate efficiently?

Location:

The locations of business units and other organizational entities such as distribution centers, service centers, call centers and payment centers. What effects will the change have on geography, people, infrastructure, data and applications? What physical facilities are needed to deploy the change?

Data:

The data and the information that are the life-fluid of the organization, flowing through the processes to accomplish the business functions. What new information content and structures are required to meet the organizational goals and objectives in alignment with the strategies defined? What data security is needed?

Application:

The IT application that enables the business processes to operate efficiently and provide decision-support information to management teams. Which applications should be developed or changed? How will the applications be integrated?

Technology:

The enabling technology that supports the operations of processes and applications. What hardware, system software and communication networks are needed to support the business? How can we leverage existing and emerging technologies?

In the primary phases of an architecture initiative of a major project, each domain described above is defined as either in-scope or out of scope. Artifacts in processes, organization and locations in POLDAT framework constitute elements of the business architecture and when are accompanied by the artifacts in data, application and technology, complete the entire enterprise architecture.

The strength and value of POLDAT framework lies in its simplicity. Organizations embarking on the first iteration of their enterprise architecture would do well to first use the POLDAT framework and then move on to a more comprehensive approach if appropriate.

2.1.2 Conducting Feasibility Study

Feasibility study addresses either a business issue to be resolved or business opportunity to be leveraged. Feasibility study conducts assessment of the potential of possible solution options being considered to satisfy the business needs in terms of financial, functional and technical feasibility. It provides the executives with a platform to develop:

- Viability of an idea for a new business opportunity,
- Strategic goals
- Balancing analysis using total cost estimate of different alternatives in order to reach an optimal solution
- Investment path

2.1.2.1 Feasibility Analysis Techniques

Feasibility consists of four major activities:

1. Conduct the current state assessment
2. Plan the feasibility efforts
3. Identify solution options
4. Assess the feasibility of each solution option

And uses the following techniques:

- Organizational chart
- Data flow diagrams
- Technology architecture diagrams
- Process flow diagrams
- Fish-bone diagram (root cause analysis)
- Work break down structure (WBS)
- Business process re-engineering
- Six Sigma
- Market surveys
- Prototyping
- Competitive analysis
- Environmental impact analysis
- Technology advancement analysis
- Cost-benefit analysis
- Pareto diagram
- Decision tables
- Structured problem solving techniques
- Probability analysis

The feasibility study report is typically comprised of the following information:

- Executive Summary
- Business problem and/or opportunity statement
- Feasibility study requirements, including the business drivers of the initiative
- For each option that was assessed, the results of the study including:
- A complete description of the solution option
- A complete description of the assessment process and methods
- A complete description of the overall results; document expected vs. actual results, scoring, and other considerations
- A list of identified risks associated with the alternative.
- A list of identified issues which adversely impact the success of the solution.
- Assumptions made during the study process to close gaps in information.
- Alternative Solution Ranking
- Ranking criteria
- Ranking scores
- Results – recommended solution, including rationale for the decision
- Appendix containing all supporting information

2.1.3 Determining Project Scope

Once the feasibility of each solution options is performed to identify the most feasible solution, it is defined in more details. Further elaboration helps to conceptualize and design the recommended solution in enough detail to build a business case, conducting initial risk assessment and propound a new project proposal for project portfolio management. Elaborating the scope includes:

- Describing the business environment in enough detail to provide context to the new project
- Describing the business requirements in enough detail to understand the need of the business
- Define the scope of the work performed to deliver the product, service or result to meet the business objectives to prepare time and cost estimates

Scope of the proposed solution is defined within the boundaries of the business problem and constraints and includes:

- Describing business objective
- Determining expected deliverables at high level
- Documenting business assumptions and constraints
- Building a high level statement of work to be performed

Any one with strong knowledge of the followings can develop or elaborate the project scope statement:

- Project Management knowledge areas and process groups
- Business process re-engineering concepts and techniques
- Zachman and POLDAT frameworks
- ISO standards and CMM model (capability maturity model)
- General management disciplines such as financial management, sales and marketing, procurement, contracts, logistics, operation planning, strategic planning, health and safety practices
- Organizational leadership, motivation, negotiation and conflict management skills
- Planning, estimating and scheduling skills
- Diagramming, documenting and presentation skills
- Identifying dependencies and analytical skills

2.1.4 Writing a Business Case: Preparation for Go/No Go Decision

Business case justifies the value addition of the project to the business through a detailed cost-benefit analysis. It also includes quantitative and qualitative benefits, time-cost estimation for breakeven point, opportunities, future profit estimation etc. It is submitted to the management team and ensures that project funding is warranted through the best investment decisions.

Developing business case requires expert level skills in financial analysis, converting project activities in financial projections, financial models and technique to project cost-benefit analysis and forecasting capability.

A business case usually consists of the followings:

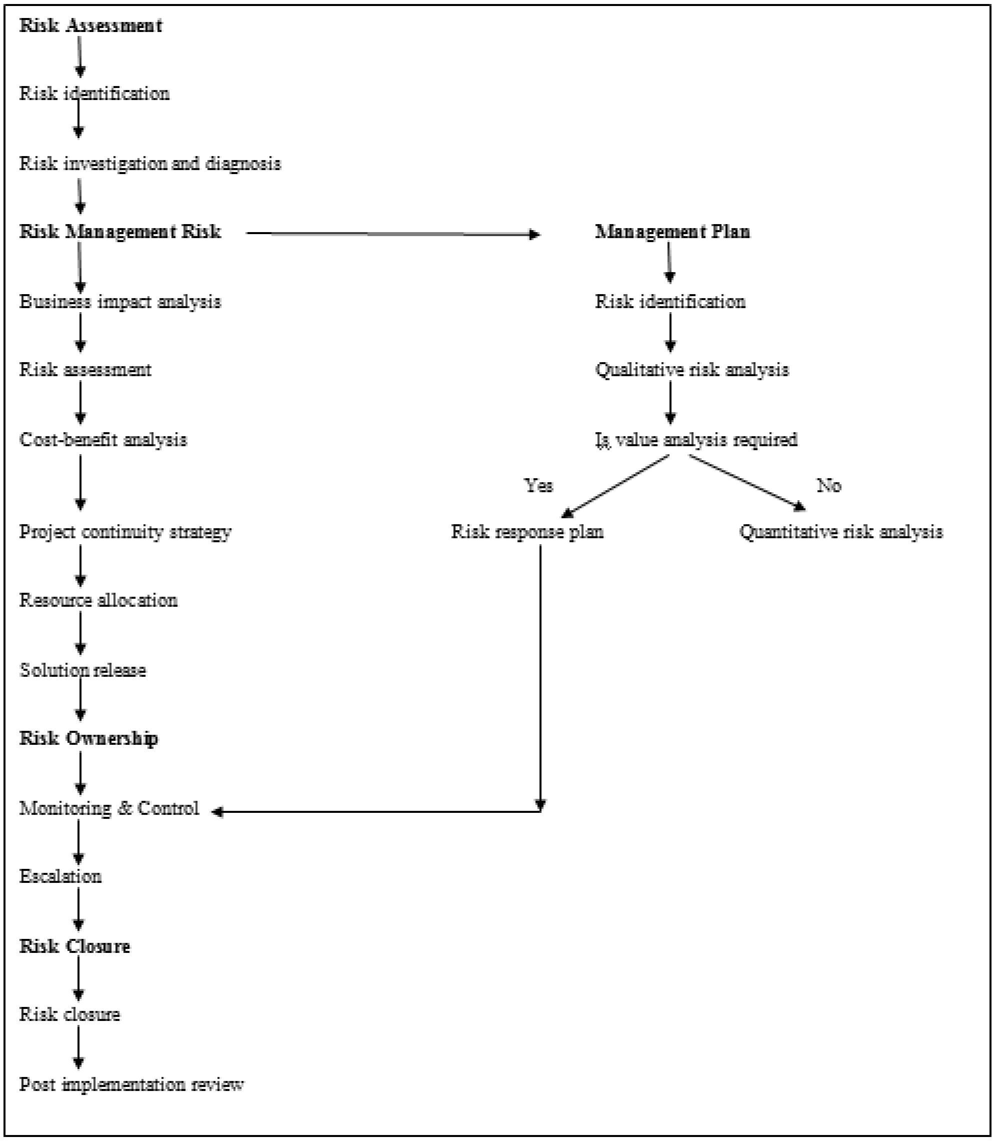
- Adequate reasoning for selected solution
- Current business process
- Problem description
- Opportunities
- Assumptions
- Constraints
- Infrastructure analysis
- Environmental analysis
- Project Objectives
- Project scope
- Out of scope
- Project costs-benefit analysis
- Performance measures
- Governance structure and model
- Accountability and operating funding model
- Project deliverables
- Change management and transition plan
- Communication management
- Risk management
- Reporting Structure
- Implementation plan
- Milestones
- Alternatives and decision analysis
- Selection criterion

Project cost-benefit analysis is done through various techniques such as Discounted cash flow, NPV (net present value), IRR (internal rate of return), ARR (average rate of return), PBP (pay back period), ABC (activity based costing) etc.

2.1.5 Initial Risk Assessment

Risk assessment is a project management knowledge area but business analyst may need to assist the project manager in initial risk assessment sometimes. Initial risk assessment is an attempt to determine if project carries more risks than the organization can bear. Project risk may have positive or negative impact on project objective, time, cost, scope or quality of the project. Risk management processes and documents are updated through out the project life cycle.

Project risk assessment needs strong knowledge of risk management concepts, change initiatives, financial analysis and forecasting, risk impact assessment, risk rating development, risk responses, financial risk analysis and technical risk analysis. Initial risk assessment process along with rest of the risk management process has been depicted in the following diagram:



The risk management process has been explained in elaboration in section 3.2.2

2.1.6 Decision Package: Go/ No Go Decision

Decision package is collection of actionable set of information to strengthen the decision of the management team. Decision package documentation refers to all the information, facts and data gathered about the proposed project. It also justifies the next steps in the process, approval of funding for next phases and proceeds with project initiation, planning and requirement development.

Preparing the decision package needs adequate skills in portfolio management and project prioritization and selection, writing recommendations and data representation.

2.1.7 Project Prioritization

Project prioritization should be done periodically to make sure that we retain our focus on the highest-priority projects. At any point of time, an organization may be running various projects concurrently. These projects must be in sync with each other under the relevant programs which must align project goals with organizational strategic objectives. Organizations functioning in an environment of uncertainty may face frequent directional changes in both organizational strategy and individual projects. If we do not reprioritize the project portfolio, we are always at the risk of pursuing the wrong projects and wasting time and resources.

Project prioritization process involves the following steps:

SWOT Analysis:

Perform a detailed analysis of strength, weakness, opportunity and threats for each project.

Create your project inventory:

1. Analyze your project portfolio to define individual projects and programs.
2. Review status of every individual project.
3. Discard projects with status "Completed" or "Failed".

Align programs and projects with current strategy:

1. Add any new projects under the appropriate program and business objective. If a new or old project doesn’t seem to fit under any current programs or business objectives, then create a “Misc.” column and file them there for now.
2. Move and/or consolidate projects as appropriate.
3. Ensure that you reflect any moves or consolidation activities on the Project Data Sheets for the respective projects.
4. The output of this step is an up-to-date Program View and Project Data Sheet (PDS) for all projects.
5. The program and project managers should do this work offline (i.e., independent from the management team).

Update the resource estimate:

Once you have updated the project data sheets for all the projects, you have current information to update the resource estimates at both project and portfolio levels.

1. Be sure every effective ongoing project has endorsement of the senior management.
2. Ensure that each of the ongoing projects has been allocated with required resources, including time, funds and HR.
3. Evaluate performance per effective project to ensure all projects run within acceptable performance levels.
4. Make an inventory of all projects that are ongoing, effective, acceptable and resourced.

Review the project details:

1. Analyze the implementation plan of every project.
2. Ensure projects are in alignment with your strategic organizational objectives.
3. Be sure projects are in alignment with each other and with the overall portfolio.
4. Make sure that in-scope, out-of-scope, project boundaries, requirements and deliverables of every project in projects inventory have been approved.
5. Estimate time span and important milestones for every project.
6. Develop a matrix that describes goals, objectives, scope and time of every project to comparative view.

Establish prioritization criteria:

Work with the C level executives to identify the criteria that will be used to prioritize the projects.

Strategic alignment
Measure all projects available in your inventory by their ability to comply with strategic goals and objectives of your company. The criteria should reflect the business strategy. They should be expressed in tangible terms. Examples are:

Relevance
Evaluate every project by the relevance to the overall portfolio.

Acceptability
Compare projects against their ability to produce acceptable deliverables.

Impact on organization
Define and measure an amount of impact every project makes to the organization and the portfolio.

Involvement
Identify the level of involvement of stakeholders involved in the project portfolio.

Quality
Evaluate projects against their ability to satisfy quality expectations defined by the customer.

Technical success
Assess the project’s probability of technical success.

Completion
Estimate the project’s probability of completion in next quarter or six months.

New business/revenue generation
Identify if project is critical to getting new business or generating additional revenue and profits.

Other criteria may include:

- Budget availability
- Total investment
- Return on investment
- Perceived business value on investment
- Payback period
- Value per dollar
- Cost saved
- Financial impact
- Customer impact (both internal and external customers)
- Supply chain impact
- Regulatory impact
- Compliance impact
- Dependency impact
- Operational impact
- Organization’s ability to perform the project
- Project management capability
- Project agility
- Value to the customer
- Risk involved
- Competitive strategic advantage
- Open new market/ opportunity
- Resource capability

Develop project prioritization matrix

Although every organization has its own method of prioritizing the projects but project prioritization matrix development includes mainly two different models:

1. Financial Model
2. Scoring Model

Financial Model

Financial model includes various financial factors to evaluate the prioritization of projects. These factors include:

1. Cost benefits ratio (CBR)
2. Net present value (NPV)
3. Internal rate of return (IRR)
4. Payback period (PP)
5. Economic value added

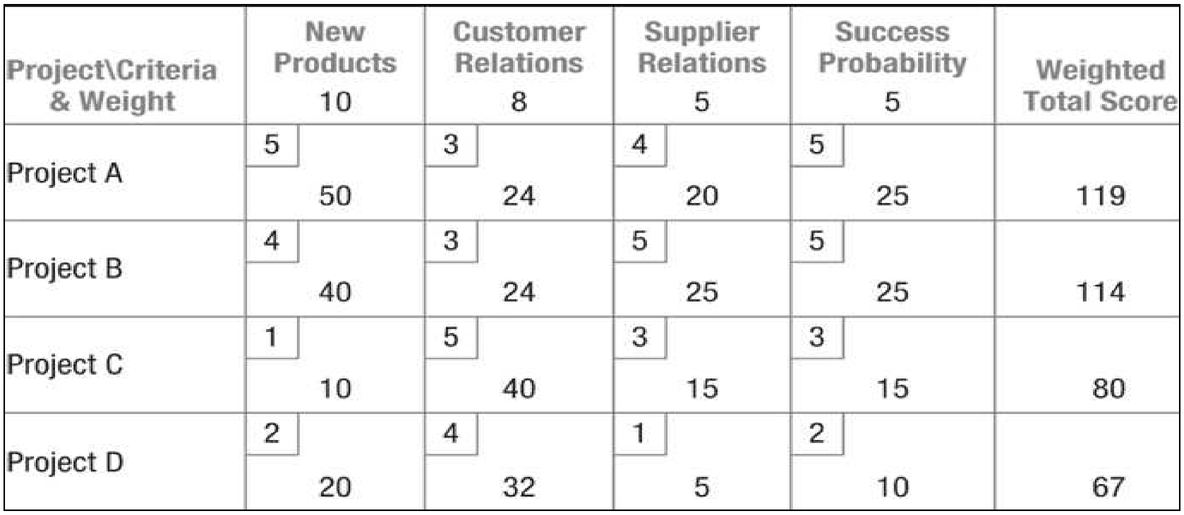
	PP	NPV	IRR	CBR
Calculation	Project cost/ Annual cash flow	Present value revenue/ present value cost	% return on project investment	Cash flow/ project investment
Neutral Result	PP = Accepted timeframe	= $0	= Cost of capital	= 1.0
Prioritization criteria	PP < Accepted timeframe	NPV > Acceptable amount	IRR > Acceptable amount	CBR > Acceptable amount

Scoring Model:

First each individual criterion is assigned a weight from 1-10 (1=lowest to 10=highest) as per its importance to the project and then each individual criterion is evaluated on the rating scale of 1 to 9, the final score of each project is a product of its weight and its rating. The scale of 1 to 5 is used to measure the rating as below:

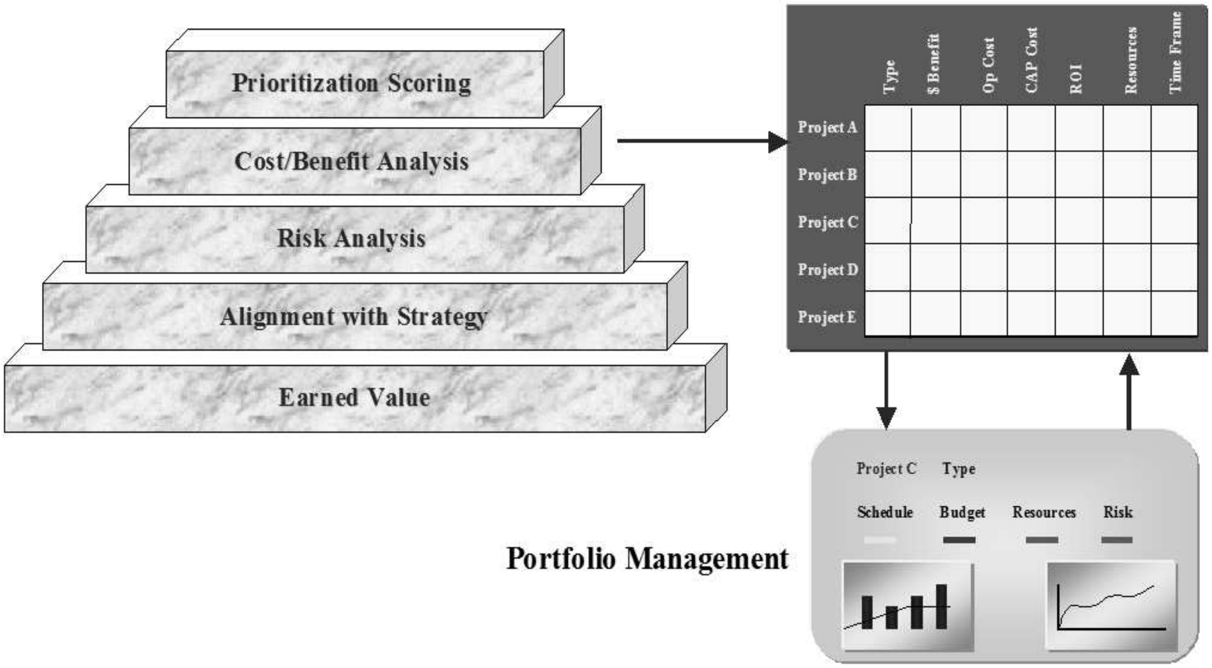
1 = High-high	2 = High-medium	3 = High-low
4= Medium-high	5 = Medium-medium	6 = Medium-low
7 = Low-high	8 = Low-medium	9 = Low-low

Below is an example of scoring model:



Project\Criteria & Weight	New Products 10	Customer Relations 8	Supplier Relations 5	Success Probability 5	Weighted Total Score
Project A	5 50	3 24	4 20	5 25	119
Project B	4 40	3 24	5 25	5 25	114
Project C	1 10	5 40	3 15	3 15	80
Project D	2 20	4 32	1 5	2 10	67

All the projects are weighted as per their final score. One thing to be noted is that in real life scenario some projects have very high project cost and negative ROI but may be categorized as "Mandatory" due to being focused on regulatory or compliance issues so they may still placed on much higher priority as compare to other projects.



2.1.8 Project Value Management

Once the projects are prioritized and initiated, project budget must be monitored and controlled through out the project life cycle to ensure the validity of the business case and continuity of project funding for the subsequent phases. Project value management is performed through project control gate review process. Project cost, schedule, risk assessment and business case are updated at every key milestone and management reviews are invited. Project priority is refined on the basis of current cost and time estimates and decision for further Go/ No go is taken by the management.

CHAPTER 3:

Requirement Gathering

3.1 Requirement Definition

In software engineering, a requirement is a description of what a system should do. Systems may have from dozens to hundreds of requirements. It describes a condition to which a system must conform; either derived directly or indirectly from user needs. A requirement for a computer system specifies what user wants or desires from a system.

SMART Requirements:

Requirements should be **S**pecific, **M**easurable, **A**ttainable, **R**ealizable and **T**raceable (SMART). Each requirement must be:

- **Unique in scope**

Is this the only requirement that defines this particular objective?

- **Precise in wording**

Are there any vague words that are difficult to interpret?

- **Bounded by concrete expectations**

Are there concrete boundaries in the objectives?

- **Irrefutably testable**

Can you build one or more test cases that will completely verify all aspects of this requirement?

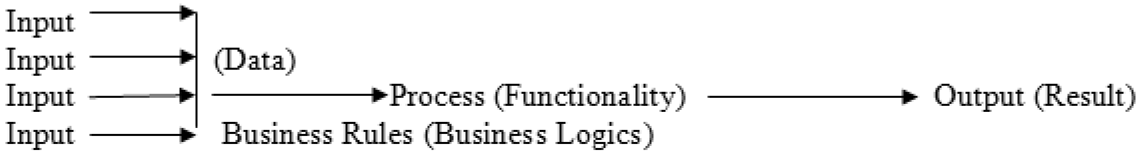
Example:

> *“The system response time shall always be reasonable for all critical transactions.”*

Revised Answer:

“The system response time for “end-to-end transactions” for “Download” shall always be “less than 5 seconds.”

In nutshell- “Requirement consists of input (data) which are governed by the business rules (business logics), processes (functionality) and output (result).”



Requirement Types

1. **Business requirement**

Business requirement defines the broad outcomes of the development of a system required by a business. It describes what purpose business desires to solve through the development of an application. Although it does not describe the design elements of the application but may describe the design and quality standards.

"What business wants or desires from a system, which you believe will deliver a business advantage to the business".

Example

Integrate the Sales, Inventory and Purchase applications to increase the level of collaboration between the three departments.

2. **User requirement**

User requirement outlines precisely what the user is expecting from this system. It describes what user wants or desires from the system which will help a user to solve certain purpose or achieve certain objectives.

Example

Integration of Sales, Inventory and Purchase applications will help:

1. Sales personnel to make accurate delivery commitment to customer while taking the sales order.
2. Inventory personnel to determine the re-order level and re-order quantity more accurately.
3. Purchase personnel to make a more accurate forecast of consumption of each material, place purchase orders only for the quantity desired and avoid over-stock/ under-stock situations to decrease the wastage of space in warehouse and wastage of material in inventory.

A good set of user requirements are needed for any project, especially computer system projects, to be successful. This is where many projects fail, in that they do not specify correctly what the system should do. In fact many systems have just been given a deadline for delivery, a budget to spend, and a vague notion of what it should do.

The root of this problem is:

- Computer systems developers rarely have as good an idea of how a business runs and should run, compared with a business user.
- Business users have little idea of what a computer system could achieve for them.
- As a result paralysis sets in and business management time is concentrated on meeting timescales and budgets, rather than what is going to be delivered.

3. Functional requirement

Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform. Functional requirements also include behavior rules, standards, policies, and other factors from the customer problem space that affect what the software needs to do to the inputs in order to provide the specified outputs.

Example

Integration of Sales, Inventory and Purchase applications must be able to:

1. Generate the real time inventory status
2. Generate the real time open order count
3. Calculate the re-order level and re-order quantity for each material
4. Forecast the consumption /sales of each material
5. Generate the reports of the periodic consumption trend of each material

4. Non-Functional requirement

Non functional requirements are the properties / qualities of the system required to meet the architectural requirement of the system in order to achieve or fulfill all other type of requirements. These emergent properties will surely be a matter of accident, not design, if the non-functional requirements, or system qualities, are not specified in advance.

In an experiment- a number of teams were given an identical set of functional requirements, but each had a different design objective: some had to make the system fast, some small to use only a small amount of computer storage, some easy to use, etc. Each team delivered a system that met their top objective fully, and other objectives to a lesser degree.

If you do not produce a set of design objectives, which are in a priority order, the developers will produce their own, and these might not be what you want. In the absence of properly defined non functional requirement, team may achieve the objective of developing a system within time, and budget but may not meet the quality required. It is very important for a system to support the business solution by its design solution to meet the quality standards and system performance requirements.

Non-functional requirements are global constraints on a software system such as development costs, operational costs, performance, reliability, maintainability, portability, robustness etc. and are known as software qualities.

Some non-functional requirement categories and trigger questions are described as below:

User Interface and Human Factors

- What type of user will be using the system?
- Will more than one type of user be using the system?
- What sort of training will be required for each type of user?
- Is it particularly important that the system be easy to learn?
- Is it particularly important that users be protected from making errors?
- What sort of input/output devices for the human interface are available and what are their characteristics?
- How system will interface other systems?

Documentation

- What kind of documentation is required?
- What audience is to be addressed by each document?

Hardware Considerations

- What hardware is the proposed system to be used on?
- What are the characteristics of the target hardware, including memory size and auxiliary storage space?

Performance Characteristics

- Are there any speed, throughput, or response time constraints on the system?
- Are there size or capacity constraints on the data to be processed by the system?
- Error Handling and Extreme Conditions
- How should the system respond to input errors?
- How should the system respond to extreme conditions?
- Workloads, response time, throughput?
- Available storage space
- Transactions per second

System Interfacing

- Is input coming from systems outside the proposed system?
- Is output going to systems outside the proposed system?
- Are there restrictions on the format or medium that must be used for input or output?

Quality Issues

- What are the requirements for reliability?
- Must the system trap faults?
- Is there a maximum acceptable time for restarting the system after a failure?
- What is the acceptable system downtime per 24-hour period?
- Is it important that the system be portable (able to move to different hardware or operating system environments)?

System Modifications

- What parts of the system are likely candidates for later modification?
- What sorts of modifications are expected?

Physical Environment

- Where will the target equipment operate?
- Will the target equipment be in one or several locations?
- Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibration and magnetic fields)?

Security Issues

- Must access to any data or the system itself be controlled?
- Is physical security an issue?
- Who can do what?

Resources and Management Issues

- How often will the system be backed up?
- Who will be responsible for the back up?
- Who is responsible for system installation?
- Who will be responsible for system maintenance?

3.2 Requirements Planning

Requirement planning involves:

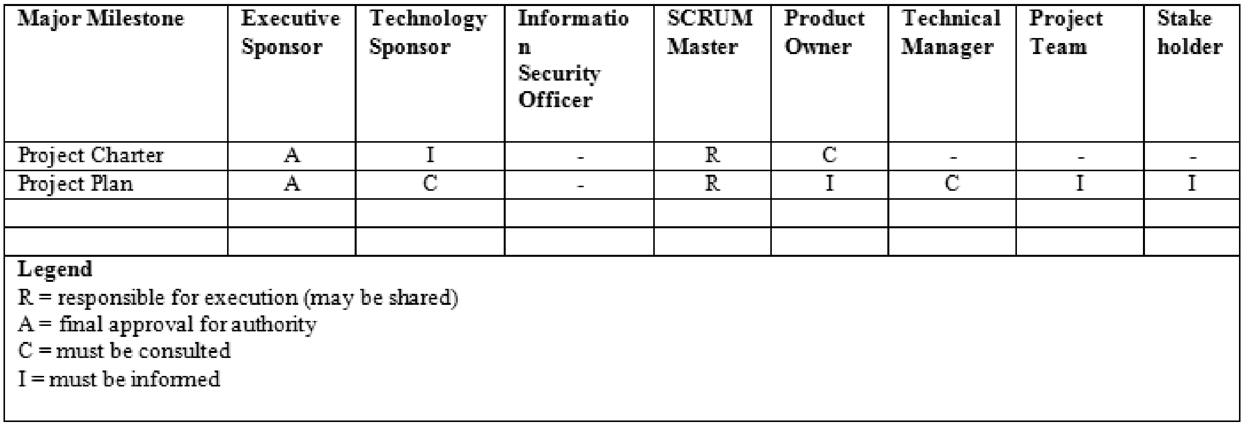
- Identification of the key roles involved in those requirements
- Defining of requirements activities that will be performed such as Requirement risk approach, managing the requirements scope, managing the requirement changes and planning the continuous communication of requirement status
- Determination of how those activities will be performed on a project
- Well planned requirements ensure the development of right deliverables and define the success path for the project. Requirement planning is usually overlooked and undervalued by most of the business analysts and is a root cause of the problems encountered in the later phases of the project

3.2.1 Identification of the key roles involved in the requirements

Key roles identification is a crucial and very essential component of requirement planning process. Key roles identification is performed by powerful tools known as RACI and POC matrix.

RACI Matrix:

RACI matrix defines the **R**esponsible, **A**ccountable, to be **C**onsulted and to be **I**nformed (RACI) roles related to a particular requirement. Following is the example of a RACI matrix:

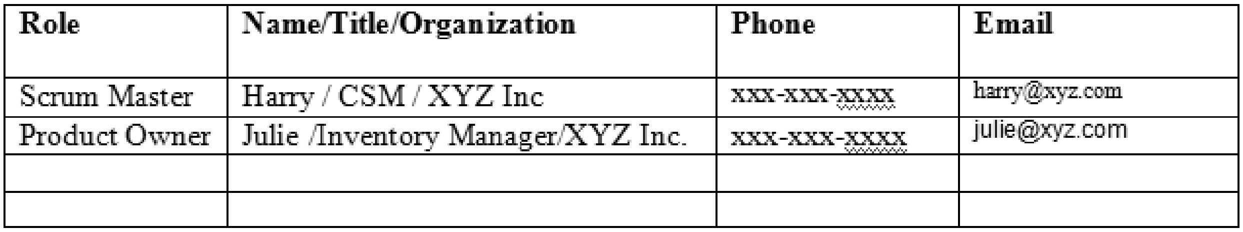


Major Milestone	Executive Sponsor	Technology Sponsor	Informatio n Security Officer	SCRUM Master	Product Owner	Technical Manager	Project Team	Stake holder
Project Charter	A	I	-	R	C	-	-	-
Project Plan	A	C	-	R	I	C	I	I

Legend
R = responsible for execution (may be shared)
A = final approval for authority
C = must be consulted
I = must be informed

POC Matrix

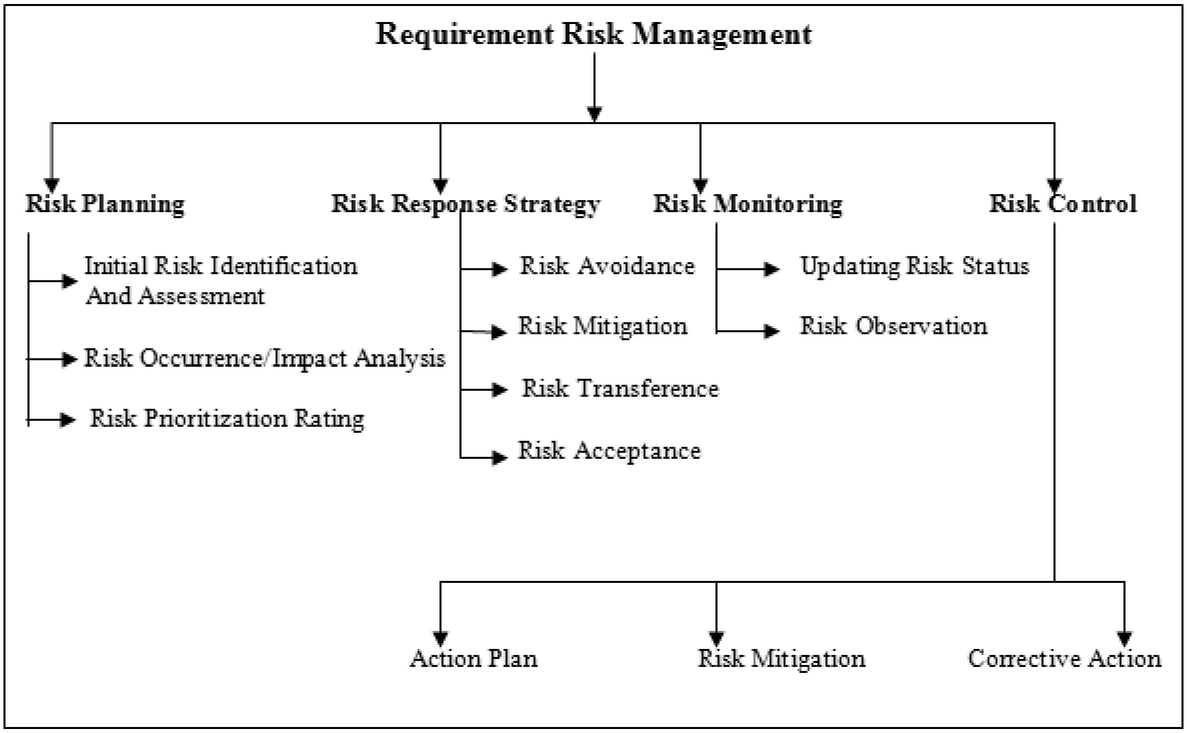
POC matrix identifies and provides the contact information for the primary and secondary contacts for the project.



Role	Name/Title/Organization	Phone	Email
Scrum Master	Harry / CSM / XYZ Inc	xxx-xxx-xxxx	harry@xyz.com
Product Owner	Julie /Inventory Manager/XYZ Inc.	xxx-xxx-xxxx	julie@xyz.com

3.2.2 Requirements Risk Management

Requirement risk and management is a subset of overall project risk and management. Most of the project risks arise due to the low or poor quality of requirements. Requirement risk management plan consists of the following activities:



Risk Management Process

A. Initial risk identification and assessment

Risk is an uncertainty about the future. Initial risk identification and assessment process identifies the requirements risks that are associated directly to specific requirements. The inclusion or addition of a risk can have a number of impacts on a project's risk profile. Requirements can also have an impact on a project's capacity to deliver on its objectives. Certain requirements may open up risks of regulatory non-compliance, legal issues, PR issues, unexpected costs or process bottlenecks and so on. In this case each requirement comes with a cost-benefit-risk profile, and each of those aspects need to be considered when analyzing requirements.

Positive Requirements Risks

Project requirements can bring a number of positive risks as well as negative risks. For example your project is changing the internal landscape of the organization, and as things change new opportunities arise.

If you are running the first successful agile project you may change the culture of project management in your organization for the better. If you deliver a new system it may be able to be used to extract non-business cased benefits.

Negative Requirements Risk

- Stakeholders are not engaged properly or early enough, and so deliver poorly articulated statements of requirements, and upon receiving requirements from the BA for validation, reject them as incomplete or incorrect.
- Changes to the regulatory or legal environment may be coming down the pipeline causing existing requirements to become obsolete.
- Requirements may be overly complex leading to a risk of poor understanding by the development team.
- Insufficient time may be allocated to requirements gathering and definition resulting in gaps or errors in requirements
- Sound requirements development practices must be supplemented with focused and proactive Requirements Risk Management (RRM). Focus is provided by knowledge of errors (root causes), defects, and failures that are likely to occur during requirements development avoiding requirements risk, preventing and detecting requirements defects, and mitigating requirements failures.

B. Risk Occurrence / Impact Analysis

Risk Occurrence:

A risk is an event that "may" occur. The probability of it occurring can range anywhere from just above 0% to just below 100%. (Note: It can't be exactly 100%, because then it would be a certainty, not a risk. And it can't be exactly 0%, or it wouldn't be a risk.)

For the probability, the following scale may be used:

1 = Very Unlikely – 0% to 5% probability
2 = Unlikely – 6% to 35% probability
3 = Likely – 36% - 65% probability
4 = Highly Likely – 66% to 95% probability
5 = Almost Certain – 96% to 100% probability

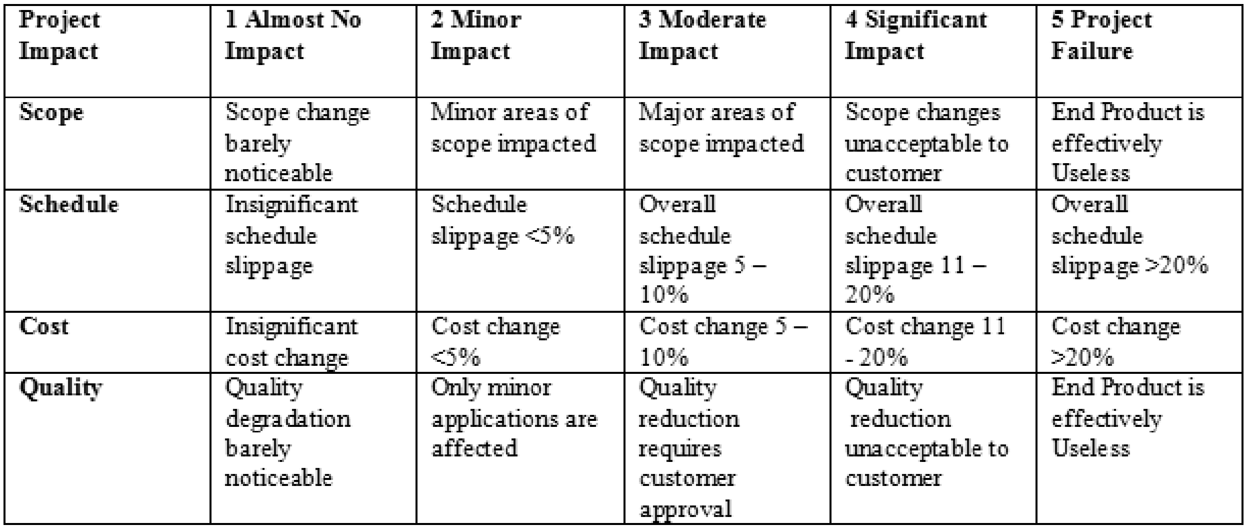
Risk Impact Analysis

A risk, by its very nature, always has a negative impact. However, the size of the impact varies in terms of cost and impact on health, human life, or some other critical factor.

For impact there is a number of different ways to look at it as risk may impact a number of different factors within the project such as costs, project schedule, lost opportunity and so on. The anticipated consequence of a risk, if it occurs, needs to be documented for those. For the impact the following scale may be used:

1 = Almost No impact on scope/cost/schedule/opportunities
2 = Minor impact on scope/cost/schedule/opportunities
3 = Moderate impact on scope/cost/schedule/opportunities
4 = Significant impact on scope/cost/schedule/opportunities
5 = Project Failure

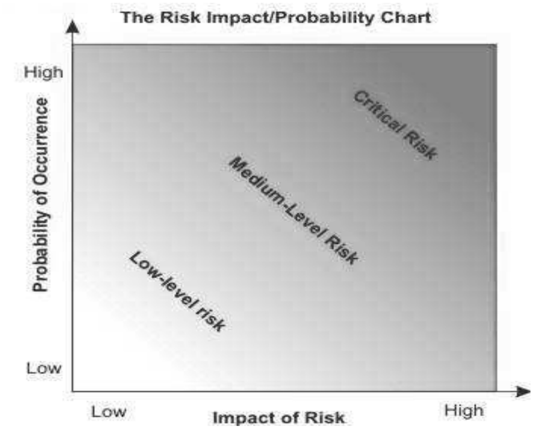
Below is an example of an impact matrix that may be used. Because all projects are unique, these factors may not fit in every case. For example a project with a mandated implementation date would have project failure with even the slightest slippage in the schedule.



Project Impact	**1 Almost No Impact**	**2 Minor Impact**	**3 Moderate Impact**	**4 Significant Impact**	**5 Project Failure**
Scope	Scope change barely noticeable	Minor areas of scope impacted	Major areas of scope impacted	Scope changes unacceptable to customer	End Product is effectively Useless
Schedule	Insignificant schedule slippage	Schedule slippage <5%	Overall schedule slippage 5 – 10%	Overall schedule slippage 11 – 20%	Overall schedule slippage >20%
Cost	Insignificant cost change	Cost change <5%	Cost change 5 – 10%	Cost change 11 - 20%	Cost change >20%
Quality	Quality degradation barely noticeable	Only minor applications are affected	Quality reduction requires customer approval	Quality reduction unacceptable to customer	End Product is effectively Useless

By multiplying the probability by the impact you determine the Risk Factor. The higher the risk factor the greater the risk to the project. The risk factor may be recorded on the Risk Management Plan template.

The Risk Impact/Probability Chart is another way of measuring these dimensions:



- List all of the likely risks that your project faces.
- Make the list as comprehensive as possible.
- Assess the probability of each risk occurring, and assign it a rating. For example, you could use a scale of 1 to 10. Assign a score of 1 when a risk is extremely unlikely to occur, and use a score of 10 when the risk is extremely likely to occur.
- Estimate the impact on the project if the risk occurs. Again, do this for each and every risk on your list. Using your 1-10 scale, assign it a 1 for little impact and a 10 for a huge, catastrophic impact.
- Map out the ratings on the Risk Impact/Probability Chart.

C. Risk Prioritization Rating:

Since you cannot manage all the risks associated with a project, you need to prioritize the risks to determine which ones should be managed. By using the risk factors you can see what risks may have the greatest impact on the project. Using this information plus any other information from the project team and stakeholders, rank the risks in priority order from highest to lowest for those risks having a significant impact on the project. Determining what risks have a significant impact on the project and should be ranked depends on the project and its ability to accept certain amounts of risk. The priority for the ranked risks may be recorded on the Risk Management Plan template.

Risk Response Strategy:

When the risks have been identified, analyzed and prioritized the next step is to determine how to respond to each risk. Within the risk response strategies there are four approaches:

Avoid: This means staying clear of the risk altogether. While avoidance obviously is the best possible course, it might not be feasible in all circumstances, e.g. the impact of the cost of avoidance might dominate the benefits of avoiding the risk. Avoidance can be accomplished by changing the process or the resources to attain an objective or sometimes modifying the objective itself to avoid the risks involved. An example of avoiding risk could be avoiding use of untested third party components in the software design, or avoiding inclusion of an inexperienced resource in the project team.

Avoidance eliminates the cause of the risk such as revising the scope to exclude that part involving the risk.

Mitigate: This means trying to reduce the probability and/or impact of the risk. Reduction in probability of occurrence would reduce the likelihood of its occurrence and reduction in impact would imply a lesser loss if the risk event occurs. 100% mitigation would be equivalent to avoidance. An example of mitigation would be an early verification of the requirements by prototyping before moving on to full fledged development.

Mitigation reduces the probability and/ or the impact of an adverse risk. This is primarily used for those risks that are to be managed by the project team.

Transfer: This implies transferring the liability of risk to a third party. While this strategy does not eliminate or mitigate the risk or its consequences itself, it transfers the responsibility of its management to someone else. Insurance is a classical example of this strategy. By buying insurance you transfer your risk to the insurance company by paying the risk premium. Fixed Cost contract is yet another example of risk transfer strategy. In a fixed cost contract the risk is transferred to the seller.

Accept: Sometimes we identify a risk but realize that time and / or resources required to formulate and enact response strategies overweigh the results of the effort. In such a case we just accept the risk. If we plan to face the occurrence as it is, it is called passive acceptance. On the other hand if we develop a contingency reserve to handle the situation if the risk occurs, we call it active acceptance.

Acceptance is accepting the risk as is and doing nothing. This is generally taken for those risks with low Risk Factors. It may be used for higher rank risks where a contingency plan is developed. If the risk occurs the contingency plan is put into operation.

Risk Response Plan

For those risks that have a response strategy of Mitigation, Acceptance, Avoidance or Transference a risk response plan needs to be developed.

Mitigation - The most common form of managing a risk is through mitigation. Within this approach a risk response plan is developed that presents the various ways the probability and/or impact of the risk may be lessened. For those risks being mitigated, the Risk Owner needs to formulate ideas as to how the risk's probability and/or impact may be reduced. These are general statements covering the various areas that may be concentrated on to lessen the risk. Action items are then developed to outline specific actions that will be taken to support those ideas in reducing the probability and impact of the risk. These action items may also be included in the project plan. A Risk Mitigation template has been depicted below to assist with the process.

Risk Mitigation Template Fields Definition

Risk Description	Enter the description of the risk as stated in the Risk Management Plan
Risk Item Identifier	Enter the risk identification information, such as Requirements #3, that was assigned to the risk in the Risk Management Plan.
Risk Priority	Enter the priority of the risk as stated in the Risk Management Plan.
Risk Factor	Enter the Risk Factor for the risk as stated in the Risk Management Plan.
Risk Response Strategy	Enter the response strategy being used for the risk (mitigation, avoidance, acceptance or transference) as indicated in the Risk Management Plan.
Risk Status	Indicate the current status of the risk; open, closed, cancelled or on-hold.
Last Updated	Enter the date when the Risk Response Plan was last updated.
Risk Owner	Enter the name of the individual who is primarily responsible for managing the risk.
Date Assigned	Record the date the risk was assigned to the Risk Owner.
Consequence if Risk Occurs	Enter a description of the impact/consequence of the risk including scope, schedule, costs, and lost opportunity.
Areas where Probability may be Reduced	List those areas that may be concentrated on to lessen the probability of the risk from occurring.
Areas where Impact may be Reduced	List those areas that may be concentrated on to lessen the impact if the risk does occur.
Attachments	If there are any attachments, please reference them here.
Action	Items within this section list all of the very specific actions that will be taken to manage this risk including how the actions will be performed and if appropriate when.

Acceptance: Because no action is taken to manage this risk the only thing that needs to be documented in the Risk Response Plan is the consequence of the risk if it occurs. No additional planning needs to be developed unless it is decided that a contingency plan will be developed. If this is the direction then the contingency plan needs to be developed and the risk must be monitored.

Avoidance: Because a change is made to the project, such as revising the scope to eliminate the risk, no Risk Response Plan needs to be developed. It is possible that the project change management process needs to be followed as a result of changes in the project.

Transference: When placing the responsibility for a risk and its consequence on someone outside the project, the project team needs to documented who will bear the responsibility and how the responsibility will be borne. This can be recorded in the consequence section of the Risk Management Plan template.

Considerations in Requirement Risk Strategy:

In order to implement a requirement risk strategy effectively, the following steps should be followed:

Requirement’s Challenge

- What’s hard and what’s easy
- What works and what doesn’t
- Why aren’t the basics enough?

Classifying Requirements Problems

- Stakeholder knowledge and behavior
- Information content and presentation
- Requirements development
- Specific problems
- Developing defect and casual profiles

Planning for RRM

- Tailor your specification strategy
- Identify risks, indicators, and root causes
- Develop, carry out, and monitor an RRM strategy

Avoiding Requirements Risk

- Reduce overall scope
- Delay risky requirements
- Stop early when success is unlikely

Preventing Requirements Defects

- Immerse stakeholders and engage experience
- Discuss and document assumptions and expectations
- Identify minimal sets of marketable features
- Use rich definitions in your glossary

Detecting Requirements Defects

- Monitor stakeholder participation
- Analyze spec style and terminology
- Review specs incrementally

Mitigating Requirements Failures

- Design tests early
- Prototype functions and interfaces
- Use incremental development

Conclusions

- Tailoring an RRM strategy
- Recording your experiences

Risk Monitoring and Controlling:

Risk monitoring and control is the process of identifying, analyzing, and planning for newly discovered risks and managing identified risks. Throughout the process, the risk owners track identified risks, reveal new risks, implement risk response plans, and gauge the risk response plans effectiveness. The key point is throughout this phase constant monitoring and due diligence is key to the success.

The objectives of risk monitoring and updating are to:

1. Systematically track the identified risks
2. Identify any new risks
3. Effectively manage the contingency reserve
4. Capture lessons learned for future risk assessment and allocation efforts

The risk monitoring and updating process occurs after the risk mitigation, planning, and allocation processes. It must continue for the life of the project because risks are dynamic. The list of risks and associated risk management strategies will likely change as the project matures and new risks develop or anticipated risks disappear.

Periodic project risk reviews repeat the tasks of identification, assessment, analysis, mitigation, planning, and allocation. Regularly scheduled project risk reviews can be used to ensure that project risk is an agenda item at all project meetings. If unanticipated risks emerge or a risk's impact is greater than expected, the planned response or risk allocation may not be adequate. At this point, the project team must perform additional response planning to control the risk.

Risk monitoring and updating tasks can vary depending on unique project goals, but three tasks should be integrated:

1. Develop consistent and comprehensive reporting procedures.
2. Monitor risk and contingency resolution.
3. Provide feedback of analysis and mitigation for future risk assessment and allocation.

The inputs to Risk Monitoring and Control are:

Risk Management Plan

The risk management plan details how to approach and manage project risk. The plan describes the how and when for monitoring risks. Additionally the Risk Management Plan provides guidance around budgeting and timing for risk-related activities, thresholds, reporting formats, and tracking.

Risk Register

The Risk Register contains the comprehensive risk listing for the project. Within this listing the key inputs into risk monitoring and control are the bought into, agreed to, realistic, and formal risk responses, the symptoms and warning signs of risk, residual and secondary risks, time and cost contingency reserves, and a watch-list of low-priority risks.

Approved Change Requests

Approved change requests are the necessary adjustments to work methods, contracts, project scope, and project schedule. Changes can impact existing risk and give rise to new risk. Approved change requests need to be reviewed from the perspective of whether they will affect risk ratings and responses of existing risks, and or if result in new risks.

Work Performance Information

Work performance information is the status of the scheduled activities being performed to accomplish the project work. When comparing the scheduled activities to the baseline, it is easy to determine whether contingency plans need to be put into place to bring the project back in line with the baseline budget and schedule. By reviewing work performance information, one can identify if trigger events have occurred, if new risk are appearing on the radar, or if identified risks are dropping from the radar.

Performance Reports

Performance reports paint a picture of the project's performance with respect to cost, scope, schedule, resources, quality, and risk. Comparing actual performance against baseline plans may unveil risks which may cause problems in the future. Performance reports use bar charts, S-curves, tables, and histograms, to organize and summarize information such as earned value analysis and project work progress.

All of these inputs help the project manager to monitoring risks and assure a successful project. Once the risk owner has gathered together all of the inputs, it is time to engage in risk monitoring and controlling. The best practices provided by PMI are:

Risk Reassessment

Risk reassessment is normally addressed at the status meetings. Throughout the project, the risk picture fluctuates: New risks arise, identified risks change, and some risks may simply disappear. To assure team members remain aware of changes in the risk picture, risks are reassessed on a regularly scheduled basis. Reassessing risks enables risk owners and the project manager to evaluate whether risk probability, impact, or urgency ratings are changing; new risks are coming into play; old risks have disappeared; and if risk responses remain adequate. If a risk's probability, impact, or urgency ratings change, or if new risks are identified, the project manager may initiate iterations of risk identification or analysis to determine the risk's effects on the project plans.

Status Meetings

Status meetings provide a forum for team members to share their experiences and inform other team members of their progress and plans. A discussion of risk should be an agenda item at every status meeting. Open collaborative discussions allow risk owners to bring to light risks which are triggering events, whether and how well the planned responses are working, and where help might be needed. Most people find it difficult to talk about risk. However, communication will become easier with practice. To assure this is the case, the project manager must encourage open discussion with no room for negative repercussions for discussing negative events.

Risk Audits

Risk audits examine and document the effectiveness of planned risk responses and their impacts on the schedule and budget. Risk audits may be scheduled activities, documented in the Project Management Plan, or they can be triggered when thresholds are exceeded. Risk audits are often performed by risk auditors, who have specialized expertise in risk assessment and auditing techniques. To ensure objectivity, risk auditors are usually not members of the project team. Some companies even bring in outside firms to perform audits.

Variance and Trend Analysis

Variance analysis examines the difference between the planned and the actual budget or schedule in order to identify unacceptable risks to the schedule, budget, quality, or scope of the project. Earned value analysis is a type of variance analysis. Trend analysis involves observing project performance over time to determine if performance is getting better or worse using a mathematical model to forecast future performance based on past results.

Technical Performance Measurement

Technical performance measurement (TPM) identifies deficiencies in meeting system requirements, provide early warning of technical problems, and monitor technical risks. The success of TPM depends upon identifying the correct key performance parameters (KPPs) at the outset of the project. KPPs are factors that measure something of importance to the project and are time/cost critical. Each KPP is linked to the work breakdown structure (WBS), and a time/cost baseline may be established for it. The project manager monitors the performance of KPPs over time and identifies variances from the plan. Variances point to risks in the project's schedule, budget, or scope.

Reserve Analysis

Reserve analysis makes a comparison of the contingency reserves to the remaining amount of risk to ascertain if there is enough reserve in the pool. Contingency reserves are buffers of time, funds, or resources set aside to handle risks that arise as a project moves forward. These risks can be anticipated, such as the risks on the Risk Register. They can be unanticipated, such as events that "come out of left field." Contingency reserves are depleted over time, as risks trigger and reserves are spent to handle them. With constraints as above monitoring the level of reserves to assure the level remains adequate to cover remaining project risk, is a necessary task.

Outputs of the Risk Monitoring and Control process are produced continually, fed into a variety of other processes. In addition, outputs of the process are used to update project and organizational documents for the benefit of future project managers. The outputs of Risk Monitoring and Control are:

Updates to the Risk Register

An updated Risk Register has the outcomes from risk assessments, audits, and risk reviews. In addition it is updated with the resulting outcome of the project risk and risk response. Was it a good response, did the response have the desired affect? The updated Risk Register is a key part of the historical record of risk management for the project and will be added to the historical archives.

Updates to Organizational Process Assets

Organizational process assets should be documented in light of the risk management processes to be used in future projects. Documents as the probability and impact matrix, risk databases, and lessons-learned information, as well as all of the project files are archived for the benefit of future project managers.

Updates to the Project Management Plan

Updates to the Project Management Plan occur if any approved changes have an impact on the risk management process. In addition, these authorized changes incur risks which are documented in the Risk Register.

Recommend Corrective Actions

Recommended corrective actions consist of two types:

Contingency plans and workaround plans. A contingency plan is a provision in the Project Management Plan that specifies how a risk will be handled if that risk occurs. The plan may be linked with money or time reserves that can be used to implement the plan. A workaround plan is a response to a negative risk that was passively accepted or not previously identified.

Recommend Preventative Actions

Recommended preventative actions assure the project follows the guidelines of the project management plan.

Requested Changes

Requested Changes are any identified changes to the project management plan. Change requests are completed and submitted to the Integrated Change Control process. All requested changes must are documented, and that approvals at the right management levels are sought and obtained.

Contingent Response Strategy:

Also known as *contingency planning*, this strategy involves development of alternatives to deal with the situation after the risk has occurred. Active acceptance of risks leads to contingency planning, whereby we anticipate a risk to occur and instead of trying to mitigate or eliminate its occurrence we plan what to do when the event occurs. Contingency reserves are commonly used tools to handle the occurrence of a risk event. Contingency reserve can imply allocation of cash, time or resources to cope with the situation when the risk event has occurred.

Fallback plans can be developed for high impact risks. A fallback plan as the name suggests, is the backup plan, in case the original contingency plan doesn’t work out as planned. An example could be identification of risk that a certain .Net programmer will resign in middle of the project. Since under the current circumstances you can do nothing to mitigate or eliminate the risk you accept it but develop a contingency plan to hire a certain programmer on hourly wages. To cope with the situation if no programmer is available on hourly wages at the time of resignation of your programmer, you develop a fall back plan of temporarily moving a software engineer from a certain low priority project to work on the assignment till an alternative can be hired.

3.2.3 Requirements Scope Management

Requirement Scope can be defined as “The work that needs to be accomplished to deliver a product, service, or result with the specified features and functions." Requirement is the smallest unit of a project, every subsequent activity is somehow and somewhere connected with a requirement in the background. A poor requirement entails to a requirement creep which further leads to project creep. A project can have a business scope creep or a technology scope creep which may cause a cost overrun. In order to control the requirement creep, scope of both functional and non-functional requirements must be managed in an efficient manner.

Requirement scope management process includes:

- Requirement Baselining
- Requirement Traceability
- Impact Identification
- Requirement Scope Change Identification
- Recording Approved Requirements

Requirement Baselining

Once all Business Requirements have been reviewed and approved, the **baseline** requirements are established. Requirement baseline is project’s fixed set of work, including business and system requirements, milestones & schedule, total work effort/cost and deliverables. This results in the creation of the first official *Business Requirements Document*.

The baseline document sets the standard for Business Requirements and is available to all Stakeholders. The document forms the basis for all work performed on the project for which it was developed for and identifies an organization's expected capabilities without limiting its potential to pursue unexpected opportunities. The baseline document also helps determine the estimated level of improvement that may be achieved through the realization of the Business Requirements.

The baseline is a dynamic document and is organic in nature. Changes to the Business Requirements Document continue through out the project and through out the life cycle of the document.

Requirement Baseline is of three types:

Functional Baseline

The functional baseline, also known as a requirements baseline, is the main product of the **Requirement phase** and is managed in accordance with the Functional Requirements Document and Data Requirements Document. It also describes where in the lifecycle the functional baseline will be established and the process by which it will be managed for this project.

Design Baseline

The design baseline reflects activities performed during the **Design Phase**. Its major component is a system/subsystem specification that defines the overall system design in terms of its subsystems, the allocation of requirements to subsystems and interfaces between subsystems and external systems. The user acceptance evaluation criteria are defined in the Verification, Validation and Test Plan. The user acceptance evaluation criteria are an important part of the design baseline. It also describes where in the lifecycle the design baseline will be established and the process by which it will be managed for this project.

Development Baseline

The development baseline, generated during the **Development Phase**, defines the detailed structure of the system being implemented. The development baseline's major components are the generation of the computer program code and the database. Other components are the training documentation, user's, operations, and maintenance documentation. It also describes where in the lifecycle the development baseline will be established and the process by which it will be managed for this project.

Product Baseline

The product baseline is established during the **Testing Phase,** specifically during the functional and performance testing. The product baseline's major component is the end system product as built by the developers. This includes the following:

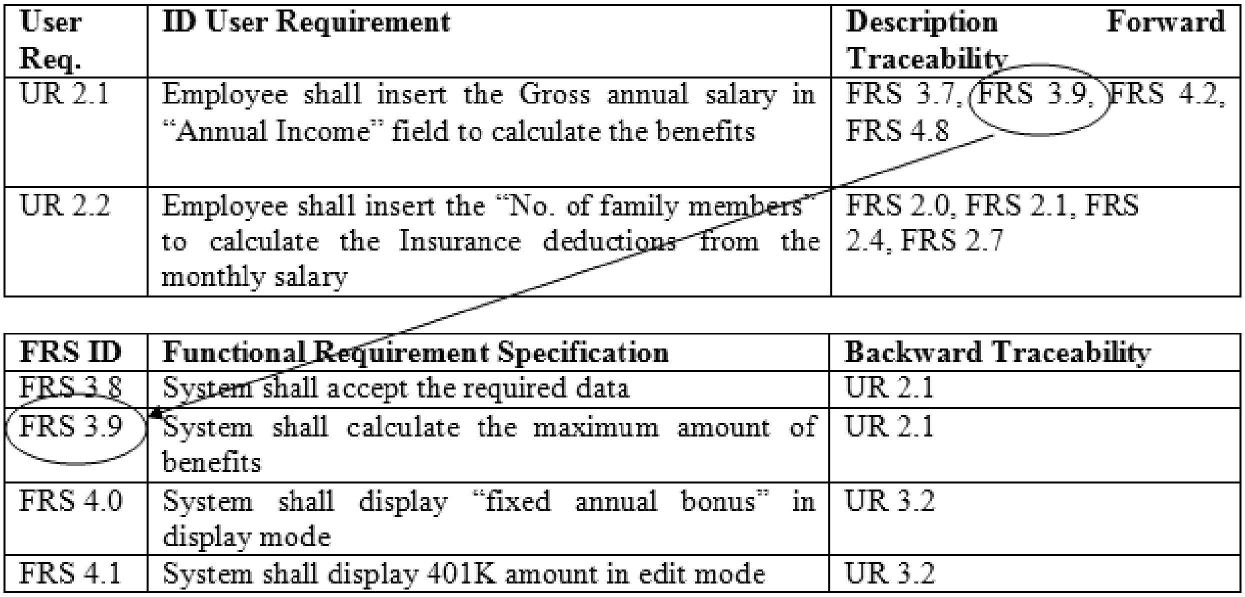
- Software
- Design and specification documentation
- Manuals (user, operations, maintenance, etc.)
- Installation and conversion procedures

Maintaining the official Business Requirements Document, right from the initial version through all the amendments, is the responsibility of Requirements Manager or the Project Manager. The baseline is reviewed through out the project to ensure scope containment. The review involves Project Manager and the Quality Management Team.

Requirement Traceability:

Requirement traceability is the ability to follow the life of a requirement, in both forwards and backwards direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases. It is a technique to manage the changes which occur after the requirements are baselined.

Tracing requirements entails documenting contextual links between the various requirements and between the work products developed to implement and verify the requirements. The requirements may include business, user, functional and test requirements. The work products include requirements documents, design specifications, software code, test plans, test cases and other artifacts of the development process. Tracing customer requirements through development and testing verifies that the customer requirements are implemented and tested. A requirements traceability matrix can simplify this process. It serves as a graphical representation of traceable relationships between requirements and work products. With a traceability matrix, IT teams can easily track customer requirements through the software development cycle, diminishing the risk of missing stated or derived requirements, especially when developing large, complex systems. A basic example of requirement traceability matrix is given below:



User Req.	ID User Requirement	Description Forward Traceability
UR 2.1	Employee shall insert the Gross annual salary in "Annual Income" field to calculate the benefits	FRS 3.7, FRS 3.9, FRS 4.2, FRS 4.8
UR 2.2	Employee shall insert the "No. of family members" to calculate the Insurance deductions from the monthly salary	FRS 2.0, FRS 2.1, FRS 2.4, FRS 2.7

FRS ID	Functional Requirement Specification	Backward Traceability
FRS 3.8	System shall accept the required data	UR 2.1
FRS 3.9	System shall calculate the maximum amount of benefits	UR 2.1
FRS 4.0	System shall display "fixed annual bonus" in display mode	UR 3.2
FRS 4.1	System shall display 401K amount in edit mode	UR 3.2

Impact Identification:

Any proposed changes must be made via a formal request and must be reviewed in relation to possible impact on requirements baseline, scope, schedule, cost, contract, risk, resources and work in progress. The requirement change may impact any or all of these areas and even an external interface to another system or a project.

Impact Identification and Analysis is performed with the help of Impact analysis report which includes:

- Relative Benefit, Cost, Risk, Priority
- Estimated man hours, Lost man hours, Cost impact (additional dollars)
- Schedule impact (days), Quality impact, other requirements affected, Other tasks affected
- Integration issues, Life cycle cost issues, other components to examine for possible changes

All the identified requirement changes, additions or removals must be documented and communicated to the stake holders. Once the approval is obtained Business Analyst must update the traceability matrix.

Requirement Scope Change Identification

Requirement scope change may occur due to various reasons such as adding new requirement, removing old in-scope requirement, modifying the current requirement, omitting a requirement, fixing a requirement error etc. Some times modifying one requirement entails to changes in other requirements, making or omitting these may also cause requirement scope creep.

Requirement scope identification is performed in alignment with the client's change control policy. A formal change control process is used to identify, evaluate, trace, report a proposed and approved requirement change. Approved requirement change is incorporated in such a way as to provide an accurate and complete audit trail.

Recording Approved Requirements

Documented record of the approved and incorporated requirements works as a GPS. It tells the current status of the project and makes it easier to figure out the direction to the destination. It also helps one to understand the stake holder's expectations and keep the project team and client on the same page. Once the change approval and control process is completed, business analyst is required to update the list of requirement and requirement traceability matrix and each updated version of these documents has to be approved. The new approved version of these documents becomes the new requirement baseline.

3.2.4 Requirements Change Management

The sole objective of developing a software business solution is to fulfill the customer's expectations by meeting the requirement of aligning the business needs with organizational objectives and to support the business activities on organizational, operational and functional level. In today's rapidly changing business scenarios, the changes in business needs are inevitable and so are the business software requirements. These requirements may change at any stage of software development life cycle or even thereafter and if not monitored and controlled effectively, they may lead to a project scope creep and change the whole direction of the project.

Customer's vision about requirements becomes more clear as they proceed ahead in the process. Each requirement change can affect not only the time, cost, quality of the software; it may lead to the change in other related requirements too resulting in hiking the time and cost of the project even further. Unmanaged or poorly managed requirement changes are the first step towards project disaster.

Requirements change management is a process of controlling and managing the proposed requests to alter the base-lined requirements performing the feasibility of alteration by measuring the level of impact and providing the recommendations accordingly. These change requests may arise due to different reasons and various sources such as "additional functional capabilities" in the software. Requirements change management process involves the following steps:

Requirements Change Identification and Documentation

Requirement changes can be identified and proposed by any team member or the customer. These changes can be related to the technical or functional capabilities of the software, business processes, technology landscape, documentations etc. These requirement changes are recorded in the requirement change request form and are proposed to the change control board (CCB).

Requirements Change Assessment and Analysis

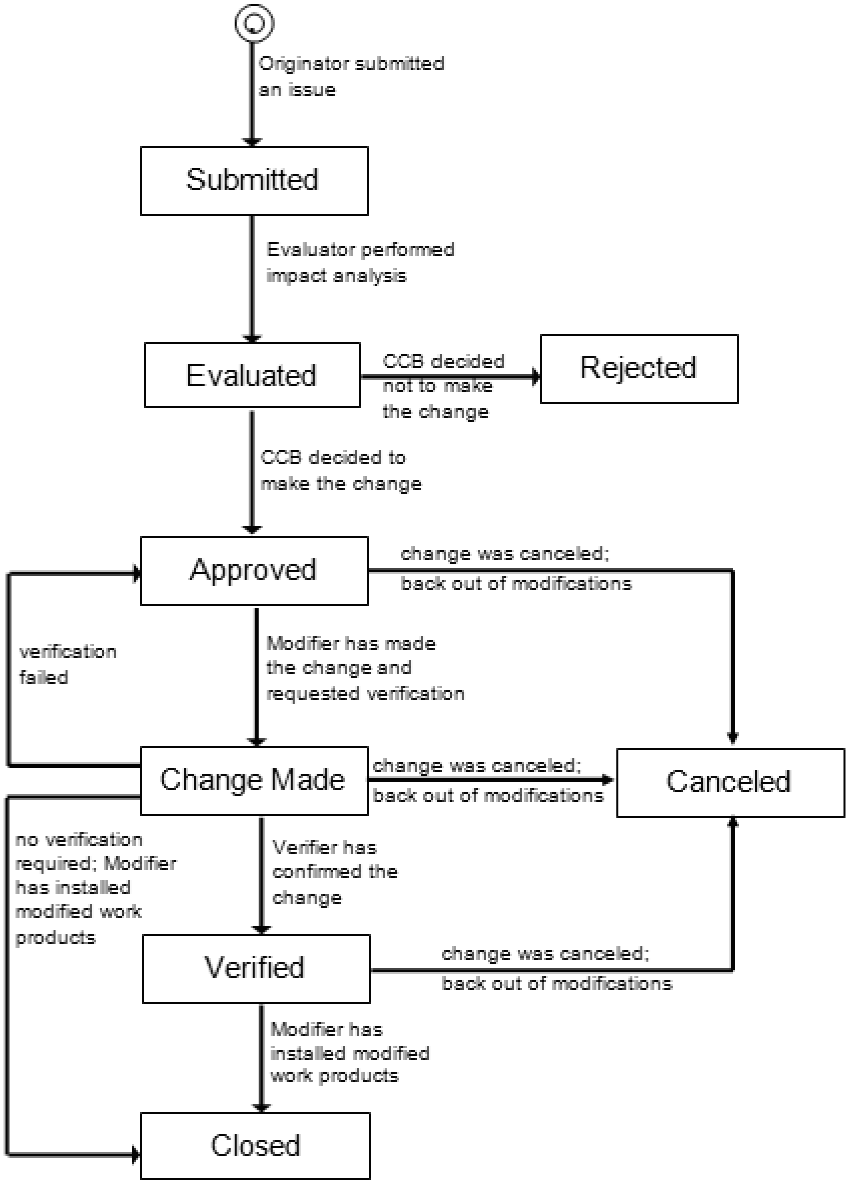
Proposed requirements changes are assessed for completeness and a detailed requirement change impact analysis is performed and severity level (critical, high, medium, low) of impact is measured. A requirement change may impact the project scope, timeline, budget, quality etc.

Requirements Change Approval

Once the impact analysis is done, it is used to propose the recommendation and requirement change request is assigned a status (submitted, verified, approved, rejected or closed). If it is approved, it may be sent to the PMO or “C” level for further approval taking their further input if necessary.

Determines the final action

A decision is documented through the change request analysis form and is communicated to PMO or project executives and team lead. Team lead further communicates it to the appropriate business analyst who documents the updated/ modified requirement, updates the change request tracking document or change request tracking system and requirement traceability matrix along with the scope document accordingly and the updated scope document reflects the new scope.


Originator submitted an issue
Submitted
Evaluator performed impact analysis
Evaluated
CCB decided not to make the change
Rejected
CCB decided to make the change
Approved
change was canceled; back out of modifications
verification failed
Modifier has made the change and requested verification
Change Made
change was canceled; back out of modifications
Canceled
no verification required; Modifier has installed modified work products
Verifier has confirmed the change
Verified
change was canceled; back out of modifications
Modifier has installed modified work products
Closed

A Sample of Requirement Management Process

3.3 Requirements Elicitation and Elicitation Techniques

Before I explain about the requirement elicitation techniques, I would suggest the readers going through the 3.1 (Requirement definition and types) to brush up your understanding about the requirements.

Requirement gathering is the most important task of a business analyst's job profile. A business analyst must be dexterous enough to gather a requirement and know if the requirement is a "good requirement" or not (see 3.1 SMART requirements). Requirement gathering is not only the most important but also the trickiest task for several reasons. Most of the time application users don't know what exactly they want and they only have a vague idea of their wish-list. Some times because of the lengthy and complicated business process when there is a involvement of too many users in requirement gathering process, their ideas about the exact process contradict with each other causing more confusions. In these types of situations, it is the responsibility of a business analyst to demystify the vision of the users and help them to figure out what exactly they want and what exactly they actually need. Some times user may wish to have a feature which is of very low business value or may even not be important to have. These types of unwanted features mostly remain unused or underused and increase the total cost of ownership (TCO) of the application.

It is not quite accurate to say that requirements are in the minds of clients; it would be more accurate to say that they are in the social system of client organization. They have to be invented, not captured or elicited and that invention has to be a cooperative venture involving the clients, the users and developers. The difficulties are mainly social, political and cultural and not technical.

A Standish group research report says that, 31% of all projects are cancelled before they ever get completed and nearly 53% of projects cost almost twice their original estimates. The reasons are:

- Users have trouble in explaining what they want.
- Developers have trouble in translating user requirements into working programs and databases.
- Both the business world and technology world are constantly changing.

Requirement Gathering Challenges

- Scope and Vision not clearly defined
- No requirement priority is defined as per the perceived business value
- Signed-off requirements keep changing
- New requirements get added in the middle of the project
- Lack of proper traceability of the requirements
- Users/customers are busy and not available to specify requirements
- Functionality built, rarely or never used

Ten Commandments of Requirement Gathering

To be successful at requirements gathering and to give your project an increased likelihood of success, follow these rules:

1. Don't assume you know what the customer wants, ask.
2. Involve the users from the start.
3. Define and agree the scope of the project.
4. Ensure requirements are specific, realistic and measurable.
5. Gain clarity if there is any doubt.
6. Create a clear, concise and thorough requirements document and share it with the customer.
7. Confirm your understanding of the requirements with the customer (play them back).
8. Avoid talking technology or solutions until the requirements are fully understood.
9. Get the requirements agreed with the stakeholders before the project starts.
10. Create a prototype if necessary to confirm or refine the customers' requirements.

Common Mistakes in Requirement Gathering

- Basing a solution on complex or cutting edge technology and then discovering that it cannot easily be rolled out to the "real world"
- Not prioritizing the User Requirements, for example 'must have', 'should have', 'could have' and 'would have,' known as the **MoSCoW principle**
- Not enough consultation with real users and practitioners
- Focusing on creating a solution without even understanding the real problem
- Lacking a clear understanding and making assumptions rather than asking
- Requirements gathering is about creating a clear, concise and agreed set of customer requirements that allow you to provide exactly what they are looking for.

Requirement Elicitation Techniques

There are several ways of requirement elicitations depending on the different situations but the most important ones are as below:

1. Brainstorming

Brainstorming is used in requirements elicitation to get as many ideas as possible from a group of people. Generally used to identify possible solutions to problems, and clarify details of opportunities. Brainstorming casts a wide net, identifying many different possibilities. Prioritization of those possibilities is important to finding the needles in the haystack.

2. Document Analysis

Reviewing the documentation of an existing system can help when creating AS-IS process documents, as well as driving gap analysis for scoping of migration projects. In an ideal world, we would even be reviewing the requirements that drove creation of the existing system – a starting point for documenting current requirements. Nuggets of information are often buried in existing documents that help us ask questions as part of validating requirement completeness.

3. Focus Group

A focus group is a gathering of people who are representative of the users or customers of a product to get feedback. The feedback can be gathered about needs / opportunities / problems to identify requirements, or can be gathered to validate and refine already elicited requirements. This form of market research is distinct from brainstorming in that it is a managed process with specific participants. There is danger in "following the crowd", and some people believe focus groups are at best ineffective. One risk is that we end up with the lowest common denominator features.

4. Interface Analysis

Interfaces for a software product can be human or machine. Integration with external systems and devices is just another interface. User centric design approaches are very effective at making sure that we create usable software. Interface analysis – reviewing the touch points with other external systems – is important to make sure we don't overlook requirements that aren't immediately visible to users.

5. Interview

Interviews of stakeholders and users are critical to creating the great software. Without understanding the goals and expectations of the users and stakeholders, we are very unlikely to satisfy them. We also have to recognize the perspective of each interviewee, so that we can properly weigh and address their inputs. Like a great reporter, listening is the skill that helps a great analyst to get more value from an interview than an average analyst.

6. Observation

The study of users in their natural habitats is what observation is about. By observing users, an analyst can identify a process flow, awkward steps, pain points and opportunities for improvement. Observation can be passive or active (asking questions while observing). Passive observation is better for getting feedback on a prototype (to refine requirements), where active observation is more effective at getting an understanding of an existing business process. Either approach can be used to uncover implicit requirements that otherwise might go overlooked.

7. Prototyping

Prototypes can be very effective at gathering feedback. Low fidelity prototypes can be used as an active listening tool. Often, when people can not articulate a particular need in the abstract, they can quickly assess if a design approach would address the need. Prototypes are most efficiently done with quick sketches of interfaces and storyboards. Prototypes are even being used as the "official requirements" in some situations.

8. Joint Application Design (JAD)

More commonly known as *Requirement Workshop* session, JAD can be very effective for gathering requirements. More structured than a brainstorming session, involved parties collaborate to document requirements. One way to capture the collaboration is with creation of domain-model artifacts (like static diagrams, activity diagrams). A workshop will be more effective with two analysts than with one, where a facilitator and a scribe work together.

9. Reverse Engineering

Is this a starting point or a last resort? When a migration project does not have access to sufficient documentation of the existing system, reverse engineering will identify what the system does. It will not identify what the system should do, and will not identify when the system does the wrong thing.

10. Survey

When collecting information from many people – too many to interview with budget and time constraints – a survey or questionnaire can be used. The survey can force users to select from choices, rate something ("Agree Strongly, Agree…"), or have open ended questions allowing free-form responses. Survey design is hard – questions can bias the respondents. Don't assume that you can create a survey on your own, and get meaningful insight from the results. I would expect that a well designed survey would provide qualitative guidance for characterizing the market. It should not be used for prioritization of features or requirements.

11. Storyboards

Storyboards, also termed "Presentation Scenarios", are sequences of images that show the relationship between user actions or inputs and system outputs. A typical storyboard will contain a number of images depicting features such as menus, dialogue boxes and windows. Storyboard sequences provide a platform for exploring and refining user requirements options via a static representation of the future system by showing them to potential users and members of a design team.

3.4 Requirements Verification and Validation

Once the requirement is gathered by the business analyst, the next most important step in the process is requirement verification and validation.

Requirement Verification

"Requirement verification is a set of activities that ensure if the business analyst has captured the requirement right."

It is a process of proving that each requirement has been satisfied. Verification can be done by logical argument, inspection, modeling, simulation, analysis, expert review, test or demonstration. Requirement verification ensures that the system complies with the system requirements and conforms to its design.

Common problems in requirement verification

"How do I know we've got all the requirements and haven't missed something?"

"How can I be sure I've got it all, and got it all right?"

No doubt these are some of the major questions we ask ourselves to ensure we're going to develop the highest quality system that meets the demanding needs of our clients. This has been one of the driving objectives of our approach - a systematic application of the best interviewing and modeling practices that gives the analyst the knowledge of what they need to do and the comfort that it has been accurately captured.

"How do you get the users' needs identified quickly and easily?"

Many experts recommend a Requirements Discovery Session with business representatives and an experienced Information Management analyst, using a methodology based on business data understanding. Not a conventional requirements interview, an RDS is focused on the business and should apply proven and easy to understand communication and modeling techniques.

"How do I deal with clients who can't express themselves, keep changing their minds and introduce new requirements?"

An experienced analyst knows what questions to ask using effective communication skills and easy-to understand modeling techniques. The biggest cause of changing requirements is not asking the right questions in the beginning. A simple, systematic approach leads the analyst and business users to discover all the information requirements, business rules and functionality, which results in a complete and accurate business specification the first time.

"How can we build a system when the users won't invest the time to discuss their needs?"

There's no magic bullet for this issue, but key success factors involve applying an approach that delivers results and is focused on yielding rapid and visible results for business clients. To satisfy the client, we must know what the client wants, and then can show that we have addressed those requirements.

Requirement Validation

“Requirement validation is a set of activities that ensure if the business analyst has captured the right requirement.”

It is a process of ensuring that:

1. The *set* of requirements is correct, complete, and consistent,
2. A model can be created that satisfies the requirements, and
3. A real-world solution can be built and tested to prove that it satisfies the requirements.
4. The system does what it is supposed to do in its intended environment. Validation determines the correctness and completeness of the end product, and ensures that the system will satisfy the actual needs of the stakeholders.

The easiest way to validate a requirement is to put yourself in user's shoe and ask yourself –

I as a	**(Role of the user)**
Wish to have	**(Feature/functionality of the application)**
So that I can	**(Purpose to be solved)**

If the objective you wish to achieve through that specific functionality of the system is not relevant to your user role, the requirement you have gathered is not the valid one (again, use the SMART approach)

Check list for requirement validation

To be consistent, the business requirements specification should be accurate, complete and clear. Below is a checklist which represents the attributes of a quality, standard business requirements specification.

Accurate

- Are the requirements consistent – not contradicting other requirements?
- Are any requirements in conflict with given stated assumptions or constraints (business environment, technical environment, cost, schedule, and resources)?
- Do the requirements support the stated business, system and project objectives?
- Are all activities and operations necessary? Are any identified requirements not required or out of scope?
- Are all data requirements necessary; are any requirements not required or out of scope?

Complete

- Are the goals and objectives of the system clearly and fully defined?
- Have all events and conditions been handled?
- Have all operations been specified? Are they sufficient to meet the stated system objectives?
- Have all objects and data in the activity specification been defined in the model(s)?
- Have all required definitions and rules for objects and data been defined?
- Does the specification satisfy the level of detail required by the design team?
- Have all undefined, unresolved, incomplete specifications been identified for resolution?

Clear

- Are all requirements free of implementation bias (not restricted to a specific design alternative)?
- Are all requirements precisely and concisely stated?
- Have all operations been stated in terms of their triggering events or conditions, information requirements, processing and outcomes?
- Is the terminology and prose understandable by the business client/users?
- Is there any ambiguity in any of the statements (operations, rules, definitions, etc.)?
- Have all assumptions been clearly stated?

Compliant

- Has the appropriate methodology been used?
- Do the deliverables conform to organizational standards, meet organizational process objectives, and follow industry standards?

Information Model – Object Specifications

- Have all objects been identified?
- Have all objects in the Activity Specification been specified?
- Have all data elements been identified?
- Has all data in the Activity Specification been specified?
- Have all necessary relationships been defined?
- Have all identified data elements been "used"? (at least created and read)
- Have all data items and relationships been correctly and precisely defined?
- Have all data items been accurately attributed to the correct objects (Normalization)?
- Have any Super classes and Subclasses been identified and specified?
- Have redundant or derived data items and relationships been identified (and/or eliminated)?

Functional Model – Activity Specifications

- Have all required activities been specified?
- Have all operations been correctly and precisely defined?
- Have all outcomes of each operation been specified?
- Have all standard/best practice/or identified life-cycle operations for each object been specified?
- Do all operations identify the event(s) or conditions which trigger them?
- Do all operations identify the operator (system or user)?
- Do all operations use strong, unambiguous action verbs?
- Are all specifications clear and unambiguous?
- Is the data used in the operation clearly understood?
- Do required operations use rules, formulas or conditions to qualify or define the processing of the operation?

- Do all operations specify or clearly imply an outcome?
- Has the Context Diagram been updated?
- Have all interfaces and activities in the Context Diagram been specified?

In the final analysis, using the methodological approach to business requirements gathering will enable an organization to collect, segregate, prioritize, analyze and document all the relevant informational and process needs for the application under design. This understanding of the business needs for data and process will result in useable, robust and sustainable systems that give an organization a competitive edge.

CHAPTER 4:

Documenting Requirements

4.1 Requirements Analysis and Documentation

Requirement Analysis

Requirement analysis is also known as requirement engineering. The objective of requirement analysis is to have a thorough understanding of business and users' needs and break it down into distinct requirements which are clearly defined, reviewed and agreed upon by the stake holders/ users or decision makers. The quality of the end product very much depends on the completeness and accuracy of the requirement representations. The focus of requirement analysis must be ensuring that the final system or product conforms to client needs rather than attempting to mold user expectations to fit the requirements.

Requirement analysis process begins with understanding the users' expectations from the system, categorizing those expectations into different type of requirement categories, analyzing these requirements to understand what system functionalities are required to meet users' expectations.

For example, a user expresses the following expectations from an intranet based system:

"I want the system to display the request for the report sent from my supervisor to me. This request must not be visible to any other user. I would like to have several parameters and criteria on the basis of which system will generate the report by retrieving data from several different tables, combining the data sets together and generating the report file which is essentially nothing but a data file. I should be able to send that report not only to my supervisor but also into our centralized repository server which arranges all the reports in a chronological order and displays the names of the senders and receivers. This report on centralized repository server must be downloadable by any user with the appropriate authorizations."

Now if one analyzes the above mentioned user requirement, one should be able to figure out the following required functionalities:

System must be able to:

1. Transfer data from user A to user B
2. Display the data transferred
3. Recombine the distributed data
4. Replicate the data
5. Locate the data / data sets
6. Archive the data
7. Transform the data
8. Sort the data
9. Upload the files

10. Download the files
11. Index the files
12. Define different authorization levels for different user roles

But one must also ask the unaddressed questions such as:

- How many users are expected to upload and download the files at the same time?
- What should be the ideal up load and download time?
- What type of users / roles will be authorized to see the data in the repository?
- How many reports are you expecting to be generated everyday?
- Should these reports be there in the repository forever or they must be automatically delete/expire after a certain time?
- Do you want the data to be visible in both online and offline mode?
- Should data support the synchronous / asynchronous operation?
- Should system display alerts for error messages and create a log for the errors occurred?
- Are you expecting the data to be transferred from or transferred to any external applications?
- Will any of the data be displayed or used by any other user who is external to the department?
- Will data go through any firewall server during the data transfer process?

Requirement Documentation

Requirement documentation is the process of documenting the verified and validated system functionalities in a well organized, sequential and easily understandable manner. This document is also known as Functional Requirement Document or Functional Requirement Specification (FRD/FRS).

Different companies use their own FRS templates. See the following examples:

Example 1:

Page Elements				
Element	**Type**	**Location**	**Business Rule**	**Action**
User Name	Text Field	Page	Mandatory Any Character which should be in database 6 to 20 characters	Captures the user name as input from the user
Password	Text Field	Page	Mandatory Any Character which should be in database	Captures the password as input from the user
			Must be displayed as asterisk entries	
			Must be between 6 to 12 characters	
Login	Button	Page		Navigates to Home page

Example 2:

Findmejob.com: Home Page

FRS ID	FRS Sub ID	Description	Business Rule	Action
FRS 1.0		User opens the URL www.findmejob.com in the browser	Site is only internet explorer and Mozilla Firefox compatible	Browser open the website www.findmejob.com
	FRS 1.1	Home page displays the fields – ***skill set and preferred location***	Location field does not accept numeric value	Accepts skill set and location input by the user
	FRS 1.2	Home page displays a ***Find Jobs*** button		Performs the job search and lists the jobs on the basis of the criteria used by
				the user
	FRS 1.2.1	Job listing	Must display jobs in chronological order	Job Title links to a job description and other details

Functional Requirements should include:

- Descriptions of data to be entered into the system
- Descriptions of operations performed by each screen
- Descriptions of work-flows performed by the system
- Descriptions of system reports or other outputs
- Who can enter the data into the system?
- How the system meets applicable regulatory requirement

The functional specification is designed to be read by a general audience. Readers should understand the system, but no particular technical knowledge should be required to understand the document. Functional requirements should include functions performed by specific screens, outlines of work-flows performed by the system and other business or compliance requirements the system must meet.

Interface requirements

- Field accepts numeric data entry
- Field only accepts dates before the current date
- Screen can print on-screen data to the printer

Business Requirements

- Data must be entered before a request can be approved
- Clicking the Approve Button moves the request to the Approval Workflow

- All the personnel using the system will be trained according to their user roles.

Regulatory/Compliance Requirements

- The database will have a functional audit trail
- The system will limit access to authorized users
- The spreadsheet can secure data with electronic signatures

Security Requirements

- Member of the Data Entry group can enter requests but not approve or delete requests
- Members of the Managers group can enter or approve a request, but not delete requests
- Members of the Administrators group cannot enter or approve requests, but can delete requests

4.2 Gap Analysis

Requirement gap analysis is a process of analyzing the current state of the system, users' expectations of the future state and finding the missing elements to fulfill that gap. A thorough understanding of the current and future state of the system is very important because if we do not know where we are currently, we can not figure out how to reach the destination and what it will take to reach there.

Gap analysis process is mainly focused on optimizing the business processes and maximizing the ROI on IT resources. It helps to find out the loose couplings and grey areas in the business processes and utilize them to their optimum potential with the help of software applications.

It is a technique for determining the steps to be taken in moving from a current state to a desired future state. Gap analysis consists of:

- Listing of characteristic factors (such as attributes, competencies, performance levels) of the present situation ("what is"),
- Cross listing factors required to achieve the future objectives ("what should be"),
- Highlighting the gaps that exist and need to be filled.

A good example of gap analysis is given below:

End State	Current State	Gaps/Action
Provide estimate of wait time to caller once they have been on hold for 15 seconds	Callers have no idea how long they will be waiting.	Software that will provide estimate of wait time to callers, research vendors and pricing, get approval
Respond to simple, common email questions	Emails are answered in the order they are	1. Develop Frequently Asked Questions page on website, to

within 2 business days	received, regardless of complexity.	reduce email volume 2. “Segment out” simple, common questions and respond to those first
Respond to inquiries that arrive by mail and give an estimate of response time within 15 business days	We do not track response time.	1. Work with Executive Secretariat 2. Consider purchasing tracking software
Reduce wait time at walk in offices to 15 minutes or less	Many customers wait more than 15 minutes.	1. Increase staff 2. Make more transactions accessible online, so customer does not have to appear in person

4.3 Writing Use Cases

What is a Use Case

A use case is a high level functionality of the system. It is the top level category of the system functionality and defines stake holder’s goal. The use case describes the system’s behavior under various conditions as it responds to a request from one of the stakeholders, called the *primary actor*. The primary actor initiates an interaction with the system to accomplish some goal. The system responds, protecting the interests of all the stakeholders. Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and conditions surrounding the requests. The use case collects together those different scenarios.

Each use case focuses on describing how to achieve a goal or a task. For most software projects, this means that multiple, perhaps dozens of use cases are needed to define the scope of the new system. The degree of formality of a particular software project and the stage of the project will influence the level of detail required in each use case.

Use cases should not be confused with the features of the system. One or more features (a.k.a. "system requirements") describe the functionality needed to meet a stakeholder request or user need (a.k.a. "user requirement"). Each feature can be analyzed into one or more use cases, which detail cases where an actor uses the system. Each use case should be traceable to its originating feature, which in turn should be traceable to its originating stakeholder/user request.

Each Use Case constitutes a complete list of events initiated by an Actor and it specifies the interaction that takes place between an Actor and the System. In a Use Case the system is viewed as opaque, where only the inputs, outputs, and functionality matter. A *use case* defines a goal-oriented set of interactions between external actors and the system under consideration. *Actors* are parties outside the system that interact with the system. A *primary* actor is one having a goal requiring the assistance of the system. A *secondary* actor is one from which the system needs assistance.

A use case should:

- Describe what the system shall do for the actor to achieve a particular goal.
- Include no implementation-specific language.
- Be at the appropriate level of detail.
- Not include detail regarding user interfaces and screens. This is done in user-interface design, which references the use case and its business rules.

How to write a Use Case

A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. The Use Case also includes possible variants of this sequence, e.g., alternative sequences that may also satisfy the goal, sequences that may lead to failure to complete the service due to exceptional behavior, error handling, etc. The system is treated as a "black box", and the interactions with system, including system responses, are as perceived from outside the system. Thus, use cases capture ***who*** (actor) does ***what*** (interaction) with the system, for what ***purpose*** (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behavior required of the system, bounding the scope of the system. The requirements in the SRS are each uniquely numbered so that they may be accounted for in the verification testing. These requirements should be mapped to the Use Case that satisfies them for accountability.

The *meat* of the Use Case is the text description. Although different companies may have their own customized templates to write the use cases but usually a use case template will contain the following:

Guidance for Use Case Template

Document each use case using the template shown in the Appendix. This section provides a description of each section in the use case template.

Use Case ID

Give each use case a unique integer sequence number identifier. Alternatively, use a hierarchical form: X.Y. Related use cases can be grouped in the hierarchy.

Use Case Name

State a concise, results-oriented name for the use case. These reflect the tasks the user needs to be able to accomplish using the system. Include an action verb and a noun. Some examples:

- View part number information.
- Manually mark hypertext source and establish link to target.

- Place an order for a CD with the updated software version.

Use Case History

Created By
Supply the name of the person who initially documented this use case.

Date Created
Enter the date on which the use case was initially documented.

Last Updated By
Supply the name of the person who performed the most recent update to the use case description.

Date Last Updated
Enter the date on which the use case was most recently updated.

Use Case Definition

Actors

An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks. Different actors often correspond to different user classes, or roles, identified from the customer community that will use the product. Name the actor that will be initiating this use case and any other actors who will participate in completing the use case.

Trigger

Identify the event that initiates the use case. This could be an external business event or system event that causes the use case to begin, or it could be the first step in the normal flow.

Description

Provide a brief description of the reason for and outcome of this use case, or a high-level description of the sequence of actions and the outcome of executing the use case.

Pre-conditions

List any activities that must take place, or any conditions that must be true, before the use case can be started. Number each precondition.

Post-conditions

Describe the state of the system at the conclusion of the use case execution. Number each post condition.

Normal Flow

Provide a detailed description of the user actions and system responses that will take place during execution of the use case under normal, expected conditions. This dialog sequence will ultimately lead to accomplishing the goal stated in the use case name and description. This description may be written as an answer to the hypothetical question, “How do I <accomplish the task stated in the use case name>?” This is best done as a numbered list of actions performed by the actor, alternating with responses provided by the system. The normal flow is numbered “X.0”, where “X” is the Use Case ID.

Alternative Flows

Document other, legitimate usage scenarios that can take place within this use case separately in this section. State the alternative flow, and describe any differences in the sequence of steps that take place. Number each alternative flow in the form “X.Y”, where “X” is the Use Case ID and Y is a sequence number for the alternative flow. For example, “5.3” would indicate the third alternative flow for use case number 5.

Exceptions

Describe any anticipated error conditions that could occur during execution of the use case, and define how the system is to respond to those conditions. Also, describe how the system is to respond if the use case execution fails for some unanticipated reason. If the use case results in a durable state change in a database or the outside world, state whether the change is rolled back, completed correctly, partially completed with a known state, or left in an undetermined state as a result of the exception. Number each alternative flow in the form “X.Y.E.Z”, where “X” is the Use Case ID, Y indicates the normal (0) or alternative (>0) flow during which this exception could take place, “E” indicates an exception, and “Z” is a sequence number for the exceptions. For example “5.0.E.2” would indicate the second exception for the normal flow for use case number 5.

Includes

List any other use cases that are included (“called”) by this use case. Common functionality that appears in multiple use cases can be split out into a separate use case that is included by the ones that need that common functionality.

Priority

Indicate the relative priority of implementing the functionality required to allow this use case to be executed. The priority scheme used must be the same as that used in the software requirements specification.

Frequency of Use

Estimate the number of times this use case will be performed by the actors per some appropriate unit of time.

Business Rules

List any business rules that influence this use case.

Special Requirements

Identify any additional requirements, such as nonfunctional requirements, for the use case that may need to be addressed during design or implementation. These may include performance requirements or other quality attributes.

Assumptions

List any assumptions that were made in the analysis that led to accepting this use case into the product description and writing the use case description.

Notes and Issues

List any additional comments about this use case or any remaining open issues that must be resolved. Identify who will resolve each issue, the due date, and what the resolution ultimately is.

A good and easy example of a use cases is ***Withdrawing money from ATM***. The below mentioned use case not only explains how exactly a use case is developed but also describes how to derive functional requirements from a use case:

Use Case: Withdrawing money from ATM

Id: UC-1.0

Description
User inserts the ATM card
User selects the language English
User enters his PIN
User enters the amount
User selects checking account for withdrawal
User selects NO for receipt
User press enters
User receives the money
End of use case

Level:
High level use case

Primary Actor:
ATM User

Stakeholders and Interests:
Bank

Pre-Conditions:
Machine must be working
ATM card should be working
User must have money in his account

Post Conditions:
Money is dispensed from ATM

Failure end condition:
If user is unable to receive the money

Minimal Guarantee
For Withdraw Cash (ATM Use Case), minimal guarantee could be, Customer is logged out of the ATM system. This minimum guarantee ensures that the system will ensure that no unauthorized withdrawals can be made from the ATM thus protecting the interest of the Bank Customer as well as the Bank's stakeholders.

Trigger:
ATM Machine reads the card successfully.

Main Success Scenario
ATM reads the card
ATM selects the language English
ATM validates the PIN
ATM verifies the amount
ATM withdraws money from checking account
ATM does not generate the receipt
ATM dispenses money

Extensions
Do you want to perform another transaction?

Variations
User enters the language Spanish
User selects YES for the receipt

Frequency: Until user withdraws $400 in 24 hours

Assumptions

The Bank Customer understands both English and Spanish language

Special Requirements

Performance

The ATM shall dispense cash within 15 seconds of user request

User Interface

- The ATM shall display all options and messages in English and Spanish languages.
- The height of letters displayed on the display console shall not be smaller than 0.5 inches. (Reference - Americans with Disabilities Act, Document xxx, para xxx).

Security

- The system shall display the letters of PIN numbers in a masked format when they are entered by the customer. i.e. Mask the PIN with characters such as ****. Rationale – This is to ensure that a bystander will not be able to read the PIN being entered by the customer.
- The ATM system will allow user to Cancel the transaction at any point and eject the ATM card within three seconds. Rationale – In case the customer in duress/in fear of own security he/she needs to quickly get away.
- The ATM system shall not print the customer's account number on the receipt of the transaction.

Issues

What is the maximum size of the PIN that a user can have?

The above mentioned use case can also be used to derive the functional requirement of the system:

Req UC 1.1	The system shall provide an option to withdraw money
Req UC 1.2	The system shall query the user for the amount of money
Req UC 1.3	The system shall query the user for the account type
Req UC 14	The system shall validate the amount is available in the user's account before releasing funds to the user
Req UC 1.5	The system shall validate the amount is a multiple of $20.
Req UC 1.6	The system shall debit the user's account upon withdrawal of funds
Req UC 1.7	The system shall be able to issue a specific amount of money to the user.

CHAPTER 5:

Software Development Life Cycle (SDLC) and Methodologies (SDM)

While learning various things about software development we come across many different terms being used, like *method*, *process*, *model*, *framework*, etc. In the following definitions I have tried to clear the distinction between these:

- **Methodology** - one specific collection of principles and/or practices
- **Methodology Family** - a set of alternative methods that exist alongside each other
- **Framework** - a skeleton (for methods) that must be customized/augmented before use
- **Model** - a description that must be implemented by a methodology or framework

Software Development Life Cycle (SDLC):

Software development life cycle describes the various phases of software development process and the sequence in which they are executed.

The software life cycle is typically broken into phases denoting activities such as requirements, design, programming, testing, installation, and operation and maintenance.

Software Development Methodology (SDM)

Software development methodology can be defined as a ***systematic approach to software development that defines development phases and specifies the activities, products, verification procedures, and completion criteria for each phase. It is a framework that is used to structure, plan, and control the process of developing an information system***.

"Software development models can be broadly divided in three categories- Linear model, Iterative model and the combination of the two".

There are heaps of models and many companies develop and adopt their own models as per there requirements but radically all these models have more or less the same pattern. Here I will describe the most prevalent software development methodologies starting with a basic model:

History of Software Methodologies: An Overview

1970s

- Structured programming since 1969

1980s

- Structured Systems Analysis and Design Methodology (SSADM) from 1980 onwards

1990s

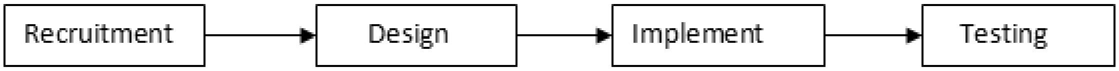
- Object-oriented programming (OOP) has been developed since the early 1960s, and developed as the dominant programming methodology during the mid-1990s.
- Rapid application development (RAD) since 1991.
- Scrum (development), since the late 1990s

2000s

- Extreme Programming since 1999
- Rational Unified Process (RUP) since 2003.
- Agile Unified Process (AUP) since 2005

Now let’s see what these different types of models are.

5.1 General Model: (Linear)



In general SDLC model requirements are gathered and converted into design. Programming code is generated during implementation and finally testing is done to confirm the application developed against the client’s requirement.

Requirement Phase:

This phase produces the deliverable in the form of Business requirement document (BRD) and Functional requirement specification (FRS). BRD consists of the information:

- Who will be using the system?
- How the system will be used?
- What data will be input to the system?
- What data will be output by the system?

FRS provides detailed information about:

- The functions system should perform
- The business logic which will process the data
- The user interface functionalities
- How the system will store or use the data
- What data will be used by the system?

Design Phase:

This phase produces the details of how the system is actually going to work. Design phase produces the deliverables in the form of Hardware and Software architecture, software application layout design and UML diagrams.

Implementation Phase:

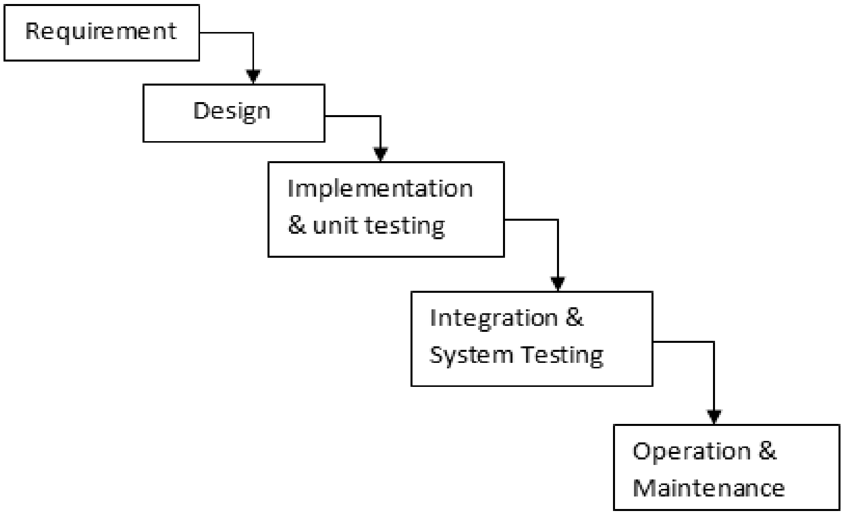
Implementation phase is the longest phase of SDLC and it may overlap with the design phase and testing phase. Implementation phase results in the program coded by the software developers.

Testing Phase:

Testing phase makes sure that the application developed confirms the business requirement gathered in Requirement phase. During this phase, testers test the application to find out the bugs.

5.2 Waterfall Model: (Linear)

Waterfall model is also known as linear sequential life cycle model. Waterfall model is a classic, most common and simplest model to understand. In this model next phase doesn't start until the previous phase is fully completed i.e. there is no overlapping of phases and there is "no go back" too. Each phase has a proper review process at the end to make sure that project is heading in the right direction and is on the right track.



Advantages

- Simple to understand and easy to use
- Rigidity of the model makes it easy to manage because:
- Each phase has specific deliverables

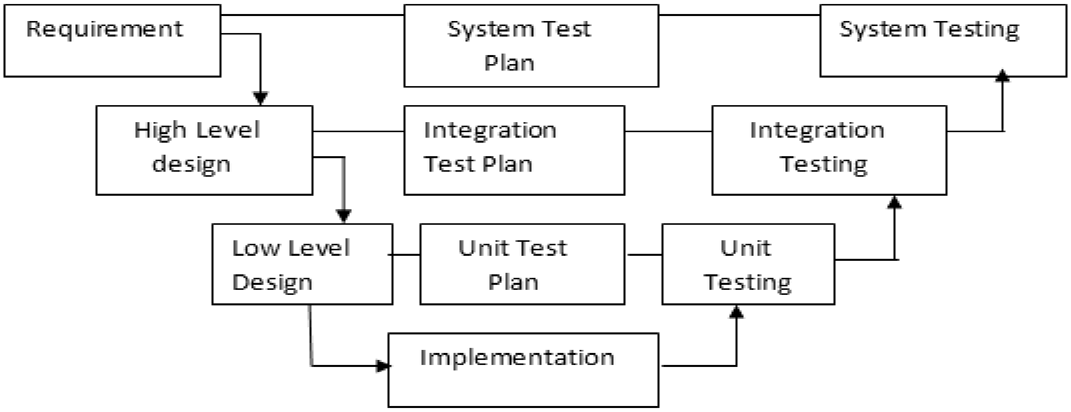
- Each phase has a proper review process
- There is no overlapping of phases.
- Works good for smaller projects where requirements are very well understood.

Disadvantages

- Adjusting scope during the life cycle can kill a project
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Poor model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Poor model where requirements are at a moderate to high risk of changing.

5.3 V-Model: (Combination)

V-Model life cycle also executes on a sequential path of processes. V model also does not have overlapping of phases. As compare to waterfall model, testing is more focused in this model. Before any coding is done, the testing procedures are developed in the early stages of life cycle.
Execution of processes starts with Requirements. Before coding begins, a system test plan is developed. The test plan ensures the compliance of functionalities with requirements. System architecture and design is kept under focus in the high-level design phase. An integration test plan is also developed in this phase. Software application and components are designed and unit tests are created in low-level design phase. The implementation phase focuses on the coding the program. Once program coding is complete, V model life cycle heads towards the right hand side of "V" and test plan execution commences.



Advantages

- Simple to understand and easy to use.
- Since the test plans are developed in the early stages of life cycle, it has greater chances of success as compare to waterfall model.

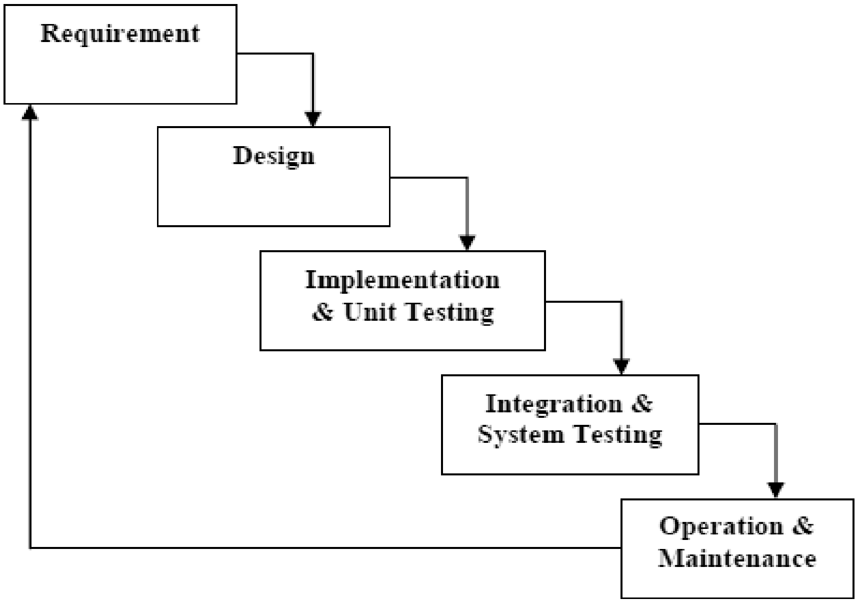
- Since the new phase can not begin until the previous phase is complete and there is no "go back", it is only good for those projects where requirements are already understood not likely to have too much changes.

Disadvantages

- It is a rigid model so adjusting scope is difficult and expensive.
- No early prototype of the software is produced because actual coding begins during the implementation phase.
- V Model doesn't provide a clear path for problems found during testing phases due to its rigidity.

5.4 Incremental Model: (Combination)

The incremental model is an intuitive derivation of waterfall model. In incremental model, multiple waterfall cycles take place. Each cycle is further divided into easily managed and smaller iterations. This model produces working software in the early stage of software life cycle during the first iteration. Subsequent iterations build on the initial software produced during the first iteration.



Advantages

- Produces working software early during the software life cycle.
- An agile, not a rigid model – adjusting scope is comparatively less costly.
- Testing and debugging is much easier due to smaller iteration.
- Risk management is much easier as it is further divided into smaller iterations.

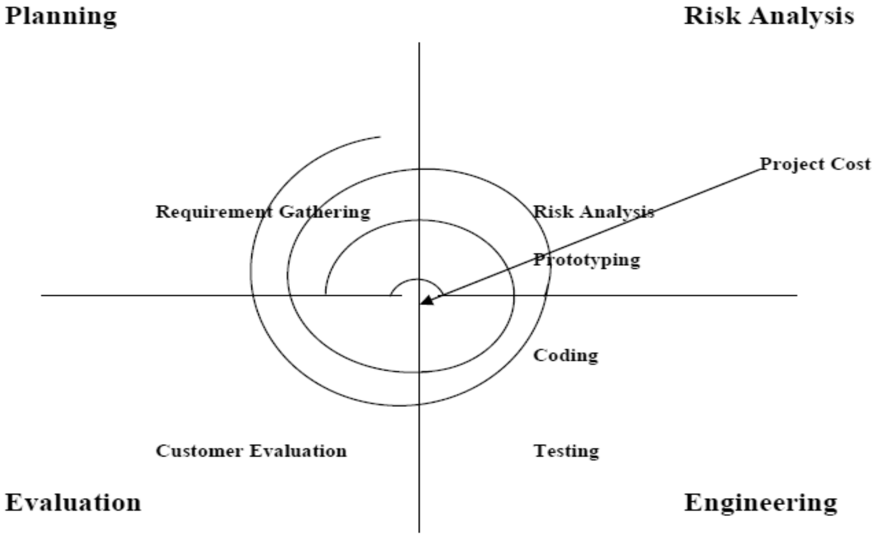
Disadvantages

- Each phase of an iteration is rigid and do not overlap each other.
- System architecture and design related problems are more likely to occur as requirements are not gathered up front for the entire software life cycle.

5.5 Spiral Model: (Combination)

The spiral model is very similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software development process repeatedly passes through these phases in iterations (called Spirals in this model). In the planning phase, requirements are gathered and risk is assessed resulting in the development of a baseline spiral. Each subsequent spiral builds on the baseline spiral. Risk identification and alternate solutions development is performed in the risk analysis phase resulting in the development of a prototype at the end of this phase. Software is produced in the engineering phase, along with testing at the end of the phase. Customers are required to evaluate the output of the project to date before the project continues to the next spiral.

The angular component denotes progress, and the radius of the spiral denotes cost in the following diagram:



Advantages

- Risk analysis is most emphasized
- It works well for long term and “must-not-fail” projects.
- In the early phases of SDLC, the software is produced.

Disadvantages

- Usually a costly model due to high amount of risk analysis.

- Lack of risk analysis expertise may result in the failure of the project.
- Not feasible for short term projects due to high costing.

5.6 Rapid Application Development (RAD): (Iterative)

Rapid Application Development (RAD) is a software development methodology that focuses on building applications in a very short amount of time; traditionally with compromises in usability, features and/or execution speed. It generically describes applications that can be designed and developed within 60-90 days, but it was originally intended to describe a process of development that involves application prototyping and iterative development.

RAD has many core elements including prototyping, iterative development, time boxing, team members, management approach, and RAD tools that make it a unique methodology.

Prototyping

A major focus of RAD is the build of a prototype with the objective of jumpstarting design and flushing out business user requirements. The objective is to build a “light-weight” version of the product in as short amount of time as possible, in days if possible. The initial version not only serves as a prototype and a proof of concept, but it also serves as a tool for refining requirements. A quick prototype development is performed with the help of Computer Aided Software Engineering (CASE) tools that focus on capturing requirements, converting them to a data model, converting the data model to a database, and generating code all in one tool.

Iterative Development

Iterative development means incrementally developing functional versions of an application in short development cycles. Each version is reviewed with the client to provide him with an opportunity to give feedback, refine requirements, and view progress. produce requirements that are input to the next version. The process is repeatedly performed until all the functionalities have been developed.

Time Boxing

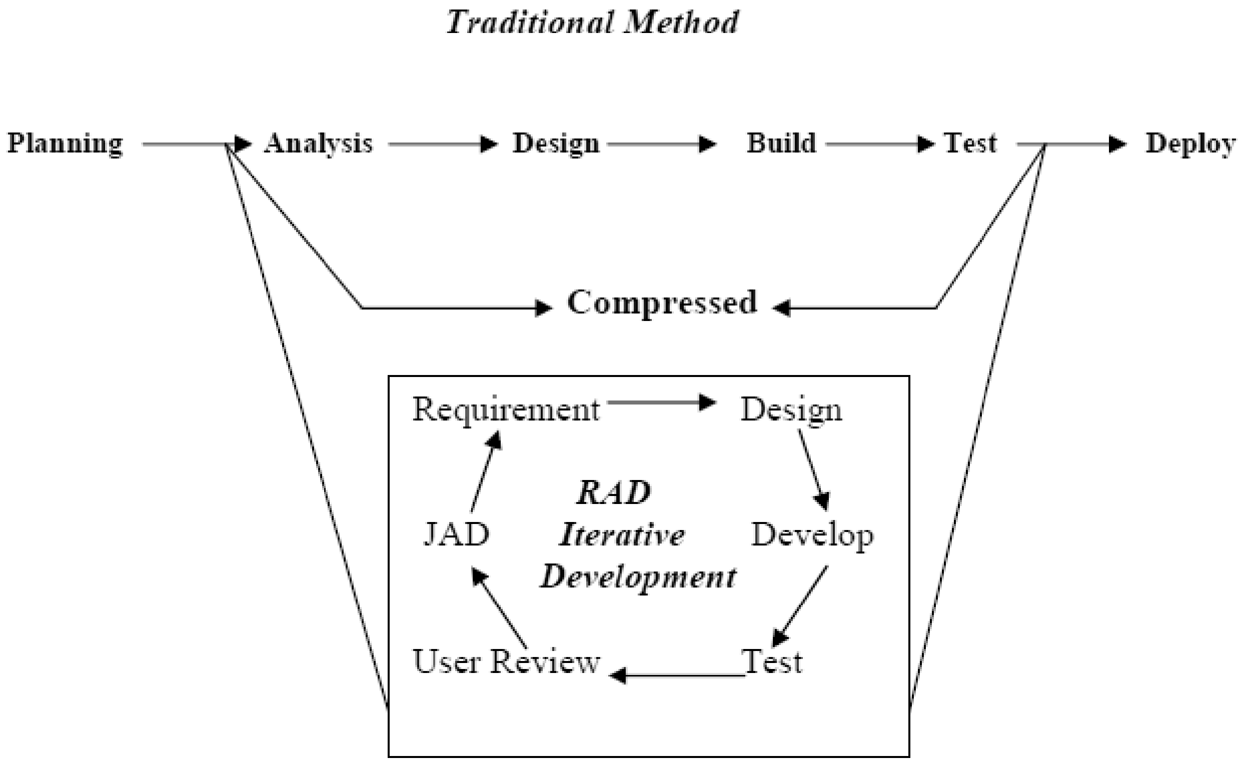
Time boxing means “First we got to make what we need now in order to save time”. It is the process of putting off functionalities to the future version of application in order to complete the current version in as short amount of time as possible.

Team Members

The RAD methodology requires small teams consisting of highly skilled, experienced, multi-facet, multitasking and motivated members. Development teams must have skilled workers with advance tools (SWAT) approach. Team members should be experienced in RAD and CASE tools.

Management Approach

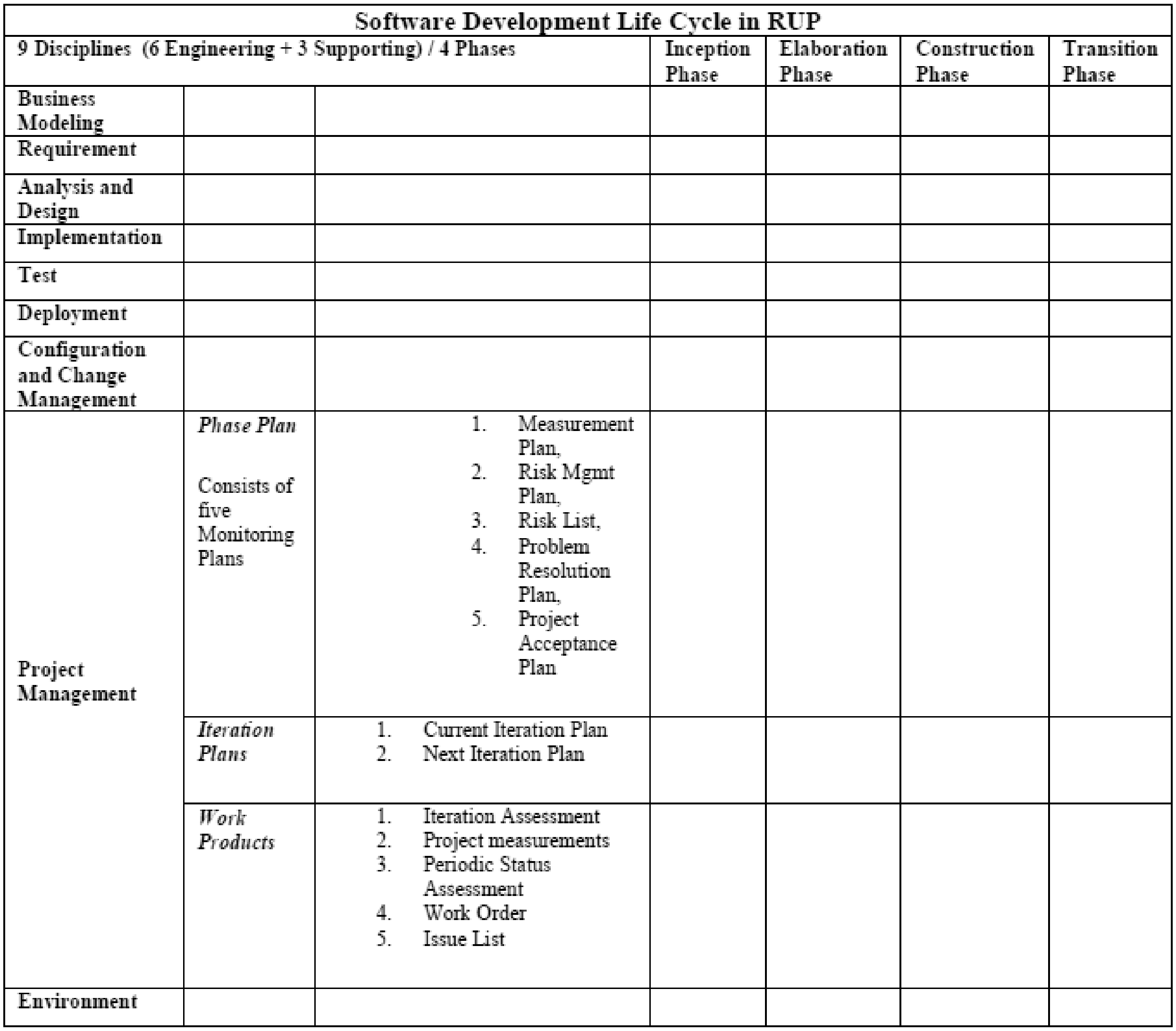
Active involvement of management plays a crucial role to minimize the risks of prolonged development processes. Management must enforce a strict and adamant time boxing.



5.7 Rational Unified Process (RUP): (Iterative)

The central focus behind developing RUP was finding the root causes of the failure of projects using rigid “waterfall" model as it does not allow individual variations of the phases. By and large the RUP process development, focused on identifying and distinguishing the attributes of failed projects. Although any project’s failure is unique in itself but each failure is a resultant of the various attributes it is not caused by a single attribute of the project. RUP has an underlying object-oriented model, using Unified Modeling Language (UML).

The structure of RUP process can be explained with the help of following diagram:

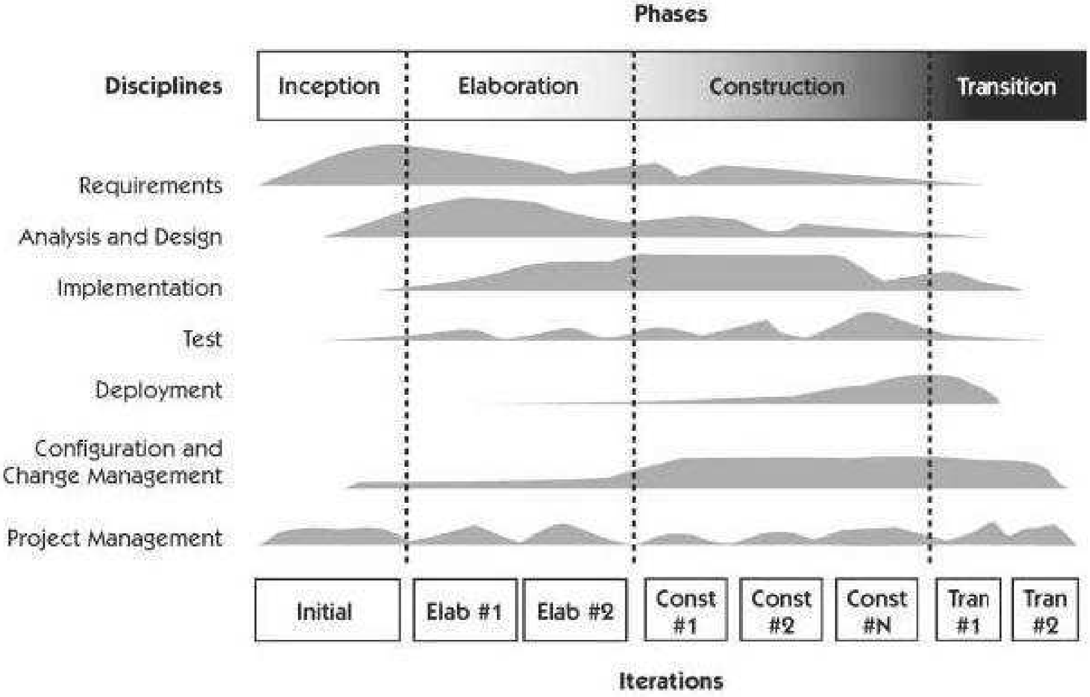


Software Development Life Cycle in RUP						
9 Disciplines (6 Engineering + 3 Supporting) / 4 Phases			Inception Phase	Elaboration Phase	Construction Phase	Transition Phase
Business Modeling						
Requirement						
Analysis and Design						
Implementation						
Test						
Deployment						
Configuration and Change Management						
Project Management	*Phase Plan* Consists of five Monitoring Plans	1. Measurement Plan, 2. Risk Mgmt Plan, 3. Risk List, 4. Problem Resolution Plan, 5. Project Acceptance Plan				
	Iteration Plans	1. Current Iteration Plan 2. Next Iteration Plan				
	Work Products	1. Iteration Assessment 2. Project measurements 3. Periodic Status Assessment 4. Work Order 5. Issue List				
Environment						

RUP building blocks

RUP is based on the following building blocks which explain what are to be produced, the necessary skills required and the step-by-step explanation of how specific development goals are achieved:

- Roles (who) – A Role defines a set of related skills, competences, and responsibilities.
- Work Products (what) – A Work Product represents something resulting from a task, including all the documents and models produced while working through the process.
- Tasks (how) – A Task describes a unit of work assigned to a Role that provides a meaningful result.



The process model for RUP

In RUP, Software Life Cycle is broken into repetitive cycles (iterations). Each iteration goes through the four phases- Inception, Elaboration, Construction and Transition. Within each iteration, the tasks are categorized into nine disciplines, six "engineering disciplines" (Business Modeling, Requirements, Analysis and Design, Implementation, Test, Deployment) and three “supporting disciplines” (Configuration and Change Management, Project Management, Environment).

Business Modeling

One of the biggest issues with process engineering is the lack of proper communication between Subject matter experts (SMEs) and software engineers and consequently the improper output of business requirement can not be shaped as a proper input to the software development effort. Business modeling discipline emphasizes on understanding the client’s organizational structure, organizational dynamics and vision and aligning the business processes, roles and responsibilities with the vision. RUP addresses this by describing how to create and maintain direct traceability between business engineering efforts and software development efforts with the help of common language known as business use cases. Business use case facilitates a commonly shared understanding among all stakeholders of what business process needs to be supported in the organization. Many projects may choose not to do business modeling.

Requirements

Requirement discipline explains how to elicit stakeholder requirements and transform them into set of work products (BRD/FRS) that can precisely scope the software application to be developed describing what the system must do. It also provides a ground for common understanding of business needs between stakeholders and software developers. Vision document and use cases are created as a part of this discipline.

Analysis and Design

A design model is the work product of this discipline which serves as a blue print of how the source code will be structured and written. This discipline also explains:

- How the functions specified in use case description will perform in a specific implementation environment
- How the system fulfils all the specified requirements
- How the system is flexible enough to adapt the functional requirement changes

Implementation

A system is realized through the implementation of components. RUP explains:

- How to organize and abstract the code to implementation of new components with well defined functionality of each component
- How to increase the reusability of the existing components to optimize the utilization of system in order to maximize the return on investment (ROI) of the business

Implementation discipline has major focus on:

- Defining the organization of the code, in terms of implementation subsystems organized in layers.
- Implementing classes and objects in terms of components (source files, binaries, executables, and others).
- Testing the developed components as units.
- Integrating the results produced by individual implementers (or teams), into an executable system.

Test Discipline

The Rational Unified Process suggests an iterative approach, which means testing continues throughout the software life cycle. This allows you to find defects as early as possible resulting in reducing the cost of fixing the defect.

Tests are carried out along four quality dimensions: *reliability, functionality, application performance, and system performance*. For each of these quality dimensions, the process describes how you go through the test life cycle of planning, design, implementation, execution and evaluation. The purposes of the Test discipline are:

- To verify the interaction between objects.
- To verify the proper integration of all components of the software.
- To verify that all requirements have been correctly implemented.
- To identify and ensure that defects are addressed prior to the deployment of the software
- Ensure that all the defects are fixed, retested and closed.

Deployment

The purpose of the deployment workflow is to successfully produce product releases, and deliver the software to its end users. It covers a wide range of activities including:

- Producing external releases of the software.
- Packaging the software.
- Distributing the software.
- Installing the software.
- Providing help and assistance to users.

In many cases, this also includes activities such as:

- Planning and conduct of beta tests.
- Migration of existing software or data.
- Formal acceptance.

Although deployment activities move around the transition phase, many of the activities need to be included in earlier phases to prepare for deployment at the end of the construction phase.

Configuration and Change Management

The Change Management discipline in RUP deals with three specific areas: configuration management, change request management, and Status and measurement management.

Configuration management is responsible for the systematic structuring of the products. Work products such as documents and models need to be under version control and these changes must be visible. It also keeps track of dependencies between artifacts so all related articles are updated when changes are made.

Change request management keeps track of the proposals for change as during the system development process many artifacts with several versions exist.

Status and measurement management Change requests have states such as new, logged, approved, assigned and complete. A change request also has attributes such as root cause, or nature (like defect and enhancement), priority etc. These states and attributes are stored in database so useful reports about the progress of the project can be produced. Change requests can be maintained with the help of rational product called ClearQuest.

Control helps avoid costly confusion, and ensures that resultant artifacts are not in conflict due to some of the following kinds of problems:

Simultaneous Update: When two or more workers work separately on the same artifact, the last one to make changes destroys the work of the former.

Limited Notification: When a problem is fixed in artifacts shared by several developers, and some of them are not notified of the change.

Multiple Versions: Most large programs are developed in evolutionary releases. One release could be in customer use, while another is in test, and the third is still in development. If problems are found in any one of the versions, fixes need to be propagated between them. Confusion can arise leading to costly fixes and re-work unless changes are carefully controlled and monitored.

Configuration and change management discipline provides guidelines for managing multiple variants of evolving software systems, tracking which versions are used in given software builds, performing builds of individual programs or entire releases according to user-defined version specifications, and enforcing site specific development policies. This explains how to manage parallel development, development done at multiple sites, and how to automate the build process. This is especially important in an iterative process where you may want to be able to do builds as often as daily, something that would become impossible without powerful automation. It also describes how to keep an audit trail on why, when and by whom any artifact was changed encompassing change request management, i.e. how to report defects, manage them through their lifecycle, and how to use defect data to track progress and trends.

Project Management

Project planning in the RUP occurs at two levels. There is a **Phase** plan which describes the entire Project and a series of fine-grained or **Iteration** plans which describe the iterations. Project management discipline emphasizes on the important aspects of an iterative development process- Risk management, Planning an iterative project, through the lifecycle and for a particular iteration, and Monitoring progress of an iterative project, metrics.

This discipline of the RUP does not attempt to cover all aspects of project management such as: Managing people (hiring, training etc.), Managing budget (defining, allocating etc.), Managing contracts with suppliers/customers etc.,

The project management discipline consists of *Plans* and *Artifacts* that are used to control the project and monitoring its performance. Such Plans are:

- The Phase Plan
- The Iteration Plan

Phase plan

Each Phase is treated as a project, controlled and measured by the **Software Development Plan** which is grouped from a subset of monitoring plans:

The Measurement Plan defines the measurement goals, the associated metrics, and the primitive metrics to be collected in the project to monitor its progress.

The Risk Management Plan details how to manage the risks associated with a project. It details the risk management tasks that will be carried out, assigned responsibilities, and any additional resources required for the risk management activity. On a smaller scale project, this plan may be embedded within the Software Development Plan.

The Risk list is a sorted list of known and open risks to the project, sorted in decreasing order of importance and associated with specific mitigation or contingency actions.

The Problem Resolution Plan describes the process used to report, analyze, and resolve problems that occur during the project.

The Product Acceptance Plan describes how the customer will evaluate the deliverable artifacts from a project to determine if they meet a predefined set of acceptance criteria. It details these acceptance criteria, and identifies the product acceptance tasks (including identification of the test cases that need to be developed) that will be carried out, and assigned responsibilities and required resources. On a smaller scale project, this plan may be embedded within the Software Development Plan.

Iteration plan

The iteration plan is a fine-grained plan with a time-sequenced set of activities and tasks, with assigned resources, containing task dependencies, for the iteration.

There are typically two iteration plans active at any point in time.

The current iteration plan is used to track progress in the current iteration.
The next iteration plan is used to plan the upcoming iteration. This plan is prepared toward the end of the current iteration.

To define the contents of iteration you need:

- The project plan
- The current status of the project (on track, late, large number of problems, requirements creep, etc.)
- A list of scenarios or use cases that must be completed by the end of the iteration
- A list of risks that must be addressed by the end of the iteration
- A list of changes that must be incorporated in the product (bug fixes, changes in requirements)

- A list of major classes or packages that must be completely implemented

These lists must be ranked. The objectives of iteration should be aggressive so that when difficulties arise, items can be dropped from the iterations based on their ranks.

Therefore there is a set of supported **Artifacts** that help in measuring and building each iteration plan.

Work Product

IBM replaced the term "artifact" with the term "work product". The work products used are:

The Iteration Assessment captures the result of an iteration, the degree to which the evaluation criteria were met, lessons learned, and changes to be done.

The project measurements are the project's active repository of metrics data. It contains the most current project, resources, process, and product measurements at the primitive and derived level.

The periodic Status Assessment provides a mechanism for managing everyone's expectations throughout the project lifecycle to ensure that the expectations of all parties are synchronized and consistent.
The work order is the Project Manager's means of communicating with the staff about what is to be done and when it is to be completed. It becomes an internal contract between the Project Manager and those assigned responsibility for completion.

The Issues List is a way to record and track problems, exceptions, anomalies, or other incomplete tasks requiring attention.

Environment

This discipline provides the software development organization with the software development tools, templates and processes which are required to equip the software development team to support the project.

5.8 Agile: (Iterative)

Agile is not a single approach to software development, it is a family of methodologies. As the name suggests Agile is “being able to quickly change direction”, it refers to a group of methodologies which focuses on the agility of the system by emphasizing on the high level of involvement from stakeholders open collaboration between team members, Iterative development and adaptability of processes. Agile methods have many things in common with RAD except RAD focuses on building a prototype at the first step whereas agile focuses on building a working release version initially.

Some of the principles behind the Agile Manifesto are:

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Even late changes in requirements are welcome
- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication (Co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

Agile team usually has 5-9 members. Each agile team contains a customer representative (CR). The CR is appointed by stakeholders to act on their behalf and be available for developers to answer mid-iteration problem-domain questions. At the end of each iteration, CR evaluates the outcome of the iteration on the basis of the business value perceived before the iteration and stakeholders review progress and re-evaluate priorities with a view to optimizing ROI and ensuring alignment of development progress with customer needs and company goals.

Agile implementation uses a formal daily face-to-face communication among team members. This specifically includes the customer representative and any interested stakeholders as observers. In a brief session, team members report to each other what they did yesterday, what they intend to do today, and what their roadblocks are. This standing face-to-face communication prevents problems being hidden. Team composition in an agile project is usually cross-functional and self-organizing without consideration for any existing corporate hierarchy or the corporate roles of team members. Team members normally take responsibility for tasks that deliver the functionality iteration requires. They decide individually how to meet iteration’s requirements. This helps minimize overall risk, and lets the project adapt to changes quickly. Agile emphasizes working software as the primary measure of progress. This, combined with the preference for face-to-face communication, produces less written documentation than other methods - though, in an agile project, documentation and other artifacts rank equally with working product. The agile method encourages stakeholders to prioritize them with other iteration outcomes based exclusively on business value perceived at the beginning of the iteration.

5.9 Scrum: (Iterative)

Scrum is iterative incremental development approach. Scrum is a set of predefined roles and practices. These roles are categorized in to categories: ***Pigs and Chickens***. Pigs are the ones who are committed to the project and if project fails “they have their bacon on the line” whereas chickens are the ones who are “involved” but not “committed” in the Scrum process. System functionalities are determined in ***Scrum meetings*** and brought into ***Sprint*** and usually come from the ***Backlog*** of the system functionalities and are documented through ***Artifacts***. Now let’s see how these roles and processes work in Scrum:

Pig Roles

Product Owner

The Product Owner represents the customer. He ensures that the Scrum team aims and shoots the "right birds". The Product Owner writes customer-centric items (typically user stories), prioritizes them and then places them in the product backlog.

Scrum Master

Scrum Master’s prime responsibility is to identify and remove the hurdles before the sprint starts so that the sprint team must encounter as little obstacles as possible in order to maintain their speed. Since the Team is self organized, the Scrum Master is not a team leader but acts as a buffer between the team and any hindering forces. The Scrum Master as facilitator of the Scrum makes sure that the Scrum process is heading in the right direction and as planned. The Scrum Master is the show-anchor who keeps the teams focused on their task and don’t let them go off the track while they sprint.

Team

The team has the responsibility to deliver the product. A team is typically made up of 5–9 people with cross-functional skills and usually consists of designers, developers, testers, technical communicators, etc.

Chicken Roles

Although chickens are only “involved” in the Scrum processes but their engagement in the project is as important as any other role. Chickens participate in the planning, development and review processes by providing feedback. The people involved as chickens are: *Business users, Stake holders and Managers*. Managers facilitate the environment for software development organizations.

Scrum Meeting

Scrum meetings are also known as “daily standup” because they are time boxed to no longer than 15 minutes and no one is allowed to sit during the scrum. It helps strictly enforce the time box. Scrum meetings start strictly on time, are conducted at the same time and same location everyday, project status is reviewed and only pigs are allowed to speak during the meeting. Every team member answers the three questions in the meeting:

- What have you done since yesterday?
- What are you planning to do by today?
- Is there any obstacle preventing you to achieve your goal?

Sprint Meeting

Sprint meetings are conducted in three phases:

Sprint Planning Meeting

Every sprint cycle lasts for 15 to 30 days. It focuses on:

- Activities to be completed during the current sprint
- Making a Sprint backlog of the activities and the expected time line
- Activities completion target for the next eight hours

Sprint Review Meeting

Sprint review meeting focuses on:

- Reviewing the activities completed/not completed after the current sprint
- Demonstrate the completed task to stakeholders

Sprint Retrospection Meeting

Sprint retrospection meeting focuses on:

- Ensuring all the team members properly retrospect the sprint completed
- Discussing what went well during the sprint completed
- Discussing the scope of improvement in the next sprint with emphasis on maintaining perpetual improvement

Artifacts

Scrum artifacts are the work products of different activities done during the Scrum. Basically there are three types of artifacts-

Product Backlog

Product backlog documents the required features prioritized by the perceived business value. This is a high level document which can be edited by anyone. Product backlog is a property of the product owner and helps him adjudging the priority of the required functionalities and expected time line.

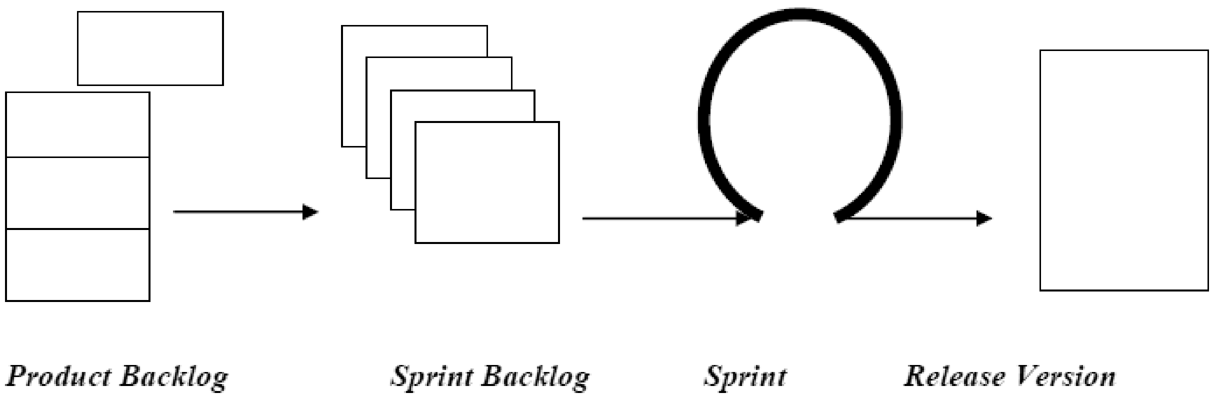
Sprint Backlog

Sprint backlog documents the features to be built in the next sprint. These features are further segregated into tasks on the basis of 4-16 hours of time line. These tasks are picked (not assigned) by the team members according to their skills and project priority. Task status is depicted on the ***task board*** as “to be done”, “under progress” or “completed”. Sprint backlog is a property of the team.

Burn-down Chart

Burn-down chart provides a “quick glance” of the sprint progress. It is updated on daily basis and displays the remaining activities of the sprint.

Following diagram shows an easy visualization of the Scrum:



5.10 Extreme Programming (XP): (Iterative)

XP is a member of agile family and emphasizes on the system's responsiveness to the client's requirements. Agile family methodologies, in general, accept the fact that "vision becomes clearer as we move ahead". Gathering all the requirement beforehand is a very unrealistic approach. On going requirement additions and changes is a part and parcel of a SDLC process and it increases the system's adaptability to requirement changes at any point of time. It also suggests that attempting to elicit all the requirements beforehand usually results in spending more time and efforts in controlling the requirements on later stages. Risky projects with dynamic requirements are perfect for XP.

2.10.1 Objective

XP organizes people to produce higher quality software more productively. In the traditional software development models, most of the requirement is gathered beforehand and thus the cost of making changes on the later stage is higher. XP attempts to reduce the cost of change by having multiple short development cycles than a long one. It strives to embrace the changes instead of "working against the changes".

2.10.2 Activities

Four basic activities have been defined in XP:

Coding: XP emphasizes most on programming as it assumes that only true product of a software development process is code. Code also helps the programmer to communicate the programming problems with each other. It can not be interpreted in more than one way.

Testing: Every piece of the code goes through the unit testing and acceptance testing multiple times to ensure the bug free software which matches the actual customer requirements.

Listening: Programmers have to listen carefully the customer's requirement and understand what business logic is required to understand the requirement. Communication between programmer and the customer plays a vital role in XP.

Designing: once the coding and testing is done, a design structure is created to organize the logic and minimizes the dependencies in the system. A good design makes sure that changing one part of the system will not affect other parts of the system.

2.10.3 Values:

Extreme programming explains five values:

Communication: XP emphasizes on shared view of system between the developers which matches the customer expectations. It recommends high level of collaboration between users and programmers, frequent verbal communication and regular feedback.

Simplicity: XP strongly suggests starting with a simplest solution by focusing on "only what we need today". Coding and designing for uncertain future needs involves the risk of spending time and resources on something which we may never need. Simple design with simple code also improves the quality of communication.

Feedback: XP involves feedback –

- From the system (continuous unit and integration testing)
- From the customer (continuous user acceptance testing)
- From the team (time estimation by the team to customer's new requirement)

Courage: Frequent requirement changes and continuous refactoring of the code produces loads of obsolete code which is unusable on a later stage. Programmers must have courage to throw the obsolete code away no matter how many efforts were made to create the code. They must also have a courage to be persistent enough by striving to solve the customer's problem.

Respect: Due to the frequent of interaction between the people, team members must have respect for each other's work. No one should feel ignored or unappreciated. This ensures high level of motivation and commitment.

2.10.4 Rules:

There are two categories of Rules in XP:

Rules of engagement:

- Business people and developers must work together
- Open progress review results as per customer's expectations and priority
- Continuous refactoring of code and team efficiency improvement
- Honest communication

- Alignment of authority and responsibility

Rules of Play:

- Continuous testing
- Quality of the code
- Common vocabulary
- Shared authority

2.10.5 Principles:

Feedback: XP explains time between an action and its feedback is very important to making changes and learning. Change may affect a part of the system which is not in the scope of a developer's task so he will not notice the flaw and this flaw is more likely to appear when the system will be in production stage. Unit test is an effective way to contribute to the speedy feedback.

Assuming Simplicity: XP assumes that reusability of the code makes it complicated. While writing the code it is to be assumed that the code will not be used again and even may be obsolete on a later stage. So having an assumption of extreme simplicity of the solution to every problem helps in understanding and communicating the solution to others and making incremental changes further on.

Embracing Change: If at one of the iteration meetings, customer's requirement changes dramatically, programmers have to embrace the change and plan the new requirement for the next iteration.

2.10.6 Practices:

XP has the various following practices:

- Pair Programming
- Planning Game
- Test Driven Development
- Whole Team
- Continuous Integration
- Refactoring or Design Improvement
- Small Releases
- Coding Standards
- Collective Code Ownership
- Simple Design
- System Metaphor
- Sustainable Pace

5.11 Dynamic System Development Methodology (DSDM): (Iterative and Incremental)

DSDM Atern is the latest version of DSDM methodology. DSDM Atern is the agile framework for management and delivery of IT and Non-IT projects. The best use of Atern is seen:

- On larger projects as it is highly scalable
- When a demonstration of a prototype is essentially required
- A large number of stake holders necessitating the need for large scale facilitated workshops

DSDM project consists of 7 phases and 9 principles. These 9 principles must confirm to the 4 values of agile manifesto.

2.11.1 7 Phases of DSDM

Pre-project phase includes project suggestion and selection of a proposed project. The pre-project phase determines if a project should be realized at all.

Feasibility Study addresses the definition of the problem, likely cost assessment and technical feasibility of delivering a system to solve business problem.

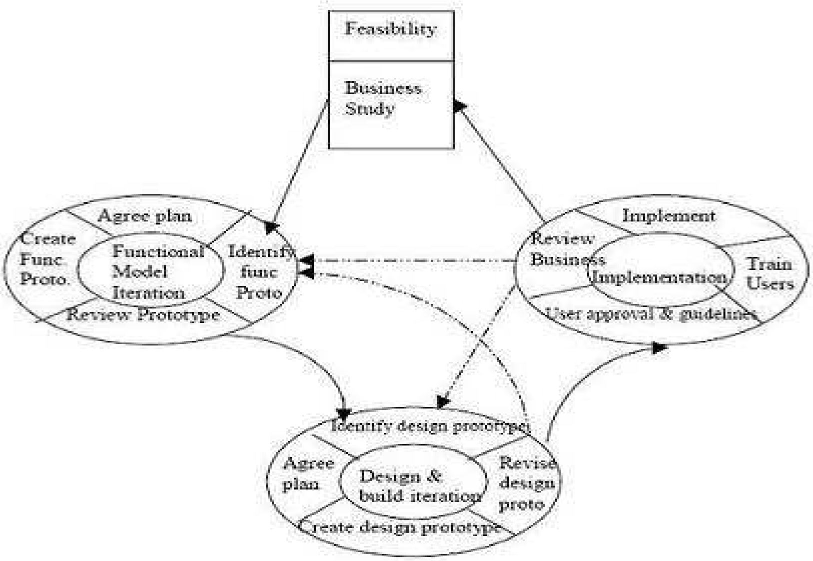
Business Study provides the platform for all subsequent work once it is decided that DSDM is an appropriate method framework.

Functional Model Iteration focuses on building the high level processing and information requirements identified during the business study i.e. it refines the business based aspects of the system.

Design and Build phase leads to the development of a high standard of system engineered to be safely placed in the hands of users.

Implementation phase encompasses the cutover from development to operation.

Post Project focuses on measuring the performance of deployed system and to check if any further enhancements are required.



2.11.2 Nine Principles

1. Active user involvement is imperative
2. Teams must be empowered to make decisions
3. Focus on frequent delivery
4. Fitness for business is criterion for accepted deliverables
5. Iterative and incremental development is mandatory
6. All changes during development must be reversible
7. Requirements are base-lined at high level\
8. Testing is integrated through out the life cycle
9. Collaborative and Co-operative approach

2.11.3 4 Values

Individual’s Importance
Agile manifesto claims the importance of “human factor”. Principles 1 and 2 confirm to this value.

Working Software
Principle 7 ensures the development of a working software as it places the business needs at crucial positions.

Collaboration
Principle 9 confirms to Collaboration value of DSDM.

Responding to Change
Principle 6 and 7 confirm to this value by welcoming and managing changes through the prioritization of requirements.

CHAPTER 6:

UML Diagrams for Business Process Modeling

UML Diagrams

The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modeling tools are wrapped under the heading of the **UML**, which stands for Unified Modeling Language™. The purpose of this course is to present important highlights of the UML.

At the center of the UML are its eight kinds of modeling diagrams, which we describe here.

1. Use case diagrams
2. Class diagrams
3. Object diagrams
4. Sequence diagrams
5. Collaboration diagrams
6. State Chart diagrams
7. Activity diagrams
8. Component and Deployment diagrams

Why is UML important?

Let's look at this question from the point of view of the construction trade. Architects design buildings. Builders use the designs to create buildings. The more complicated the building, the more critical the communication between architect and builder. Blueprints are the standard graphical language that both architects and builders must learn as part of their trade.

Writing software is not unlike constructing a building. The more complicated the underlying system, the more critical the communication among everyone involved in creating and deploying the software. In the past decade, the UML has emerged as the software blueprint language for analysts, designers, and programmers alike. It is now part of the software trade. The UML gives everyone from business analyst to designer to programmer a common vocabulary to talk about software design.

The UML is applicable to object-oriented problem solving. Anyone interested in learning UML must be familiar with the underlying tenet of object-oriented problem solving -- it all begins with the construction of a model. A **model** is an abstraction of the underlying problem. The **domain** is the actual world from which the problem comes.

Models consist of **objects** that interact by sending each other **messages**. Think of an object as "alive." Objects have things they know (**attributes**) and things they can do (**behaviors** or **operations**). The values of an object's attributes determine its **state**.

Classes are the "blueprints" for objects. A class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity. Objects are **instances** of classes.

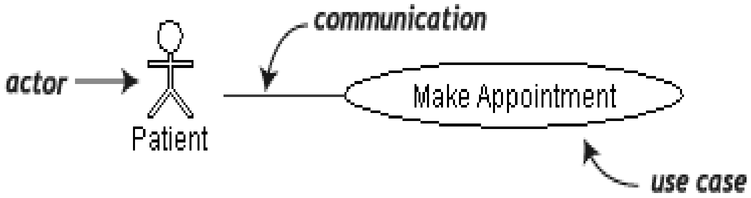
6.1 Use Case Diagrams

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on *what* a system does rather than *how*.

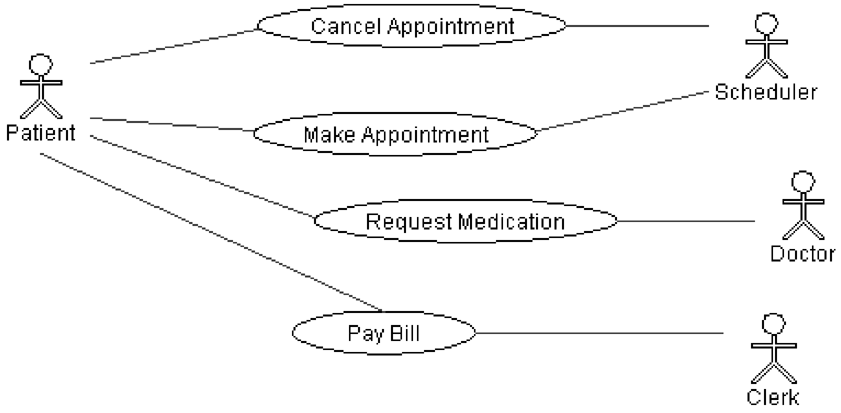
Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system. Here is a scenario for a medical clinic.

"A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot."

A **use case** is a summary of scenarios for a single task or goal. An **actor** is who or what initiates the events involved in that task. Actors are simply roles that people or objects play. The picture below is a **Make Appointment** use case for the medical clinic. The actor is a **Patient**. The connection between actor and use case is a **communication association** (or **communication** for short).



Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases. A use case diagram is a collection of actors, use cases, and their communications. We've put **Make Appointment** as part of a diagram with four actors and four use cases. Notice that a single use case can have multiple actors.

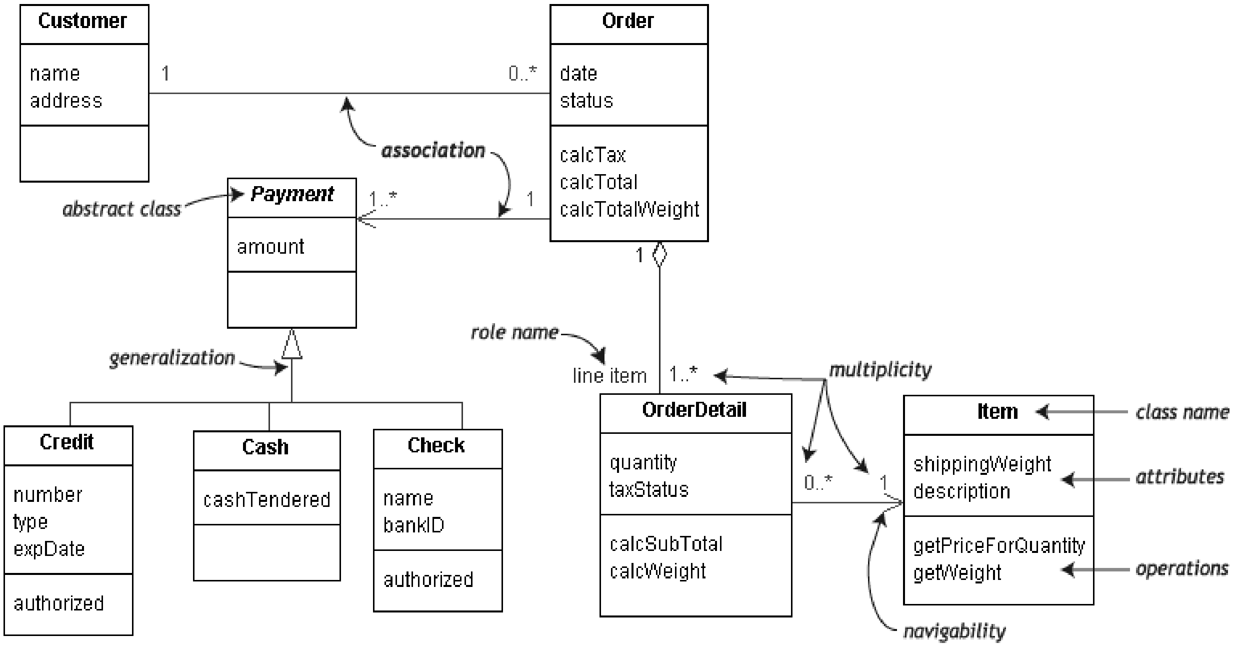


Use case diagrams are helpful in three areas.

- **Determining features (requirements)**. New use cases often generate new requirements as the system is analyzed and the design takes shape.
- **Communicating with clients**. Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- **Generating test cases**. The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.

6.2 Class Diagrams

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static -- they display what interacts but not what happens when they do interact. The class diagram below models a customer order from a retail catalog. The central class is the **Order**. Associated with it are the **Customer** making the purchase and the **Payment**. A **Payment** is one of three kinds: **Cash**, **Check**, or **Credit**. The order contains **OrderDetails** (line items), each with its associated **Item**.



UML class notation is a rectangle divided into three parts: class name, attributes, and operations. Names of abstract classes, such as ***Payment***, are in italics. Relationships between classes are the connecting links.

Our class diagram has three kinds of relationships:

Association -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.

Aggregation -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, **Order** has a collection of **OrderDetails**.

Generalization -- an inheritance link indicating one class is a super class of the other. A generalization has a triangle pointing to the super class. ***Payment*** is a super class of **Cash**, **Check**, and **Credit**.

An association has two ends. An end may have a **role name** to clarify the nature of the association. For example, an **OrderDetail** is a line item of each **Order**.

A **navigability** arrow on an association shows which direction the association can be traversed or queried. An **OrderDetail** can be queried about its **Item**, but not the other way around. The arrow also lets you know who "owns" the association's implementation; in this case, **OrderDetail** has an **Item**. Associations with no navigability arrows are bi-directional.
The **multiplicity** of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers. In our example, there can be only one **Customer** for each **Order**, but a **Customer** can have any number of **Orders**.

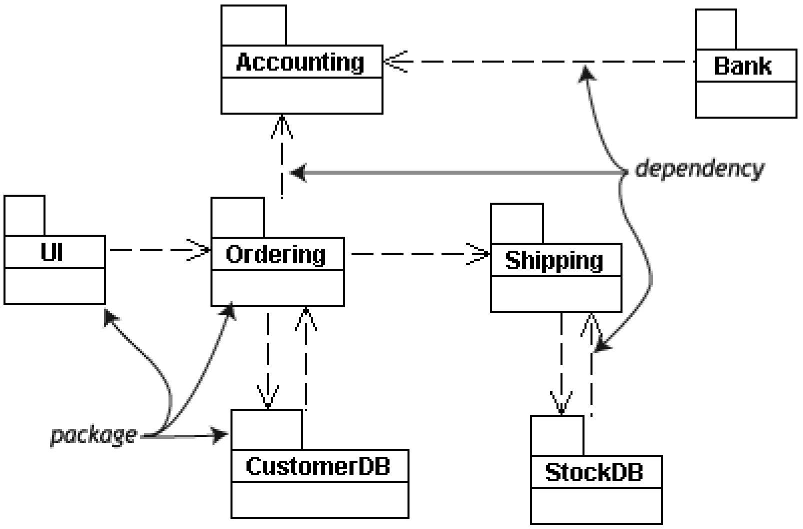
This table gives the most common multiplicities.

Multiplicities	**Meaning**
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* *****	No limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

Every class diagram has classes, associations, and multiplicities. Navigability and roles are optional items placed in a diagram to provide clarity.

6.3 Package and Object Diagrams

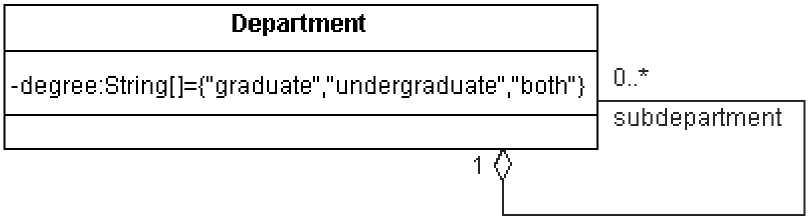
To simplify complex class diagrams, you can group classes into **packages**. A package is a collection of logically related UML elements. The diagram below is a business model in which the classes are grouped into packages.



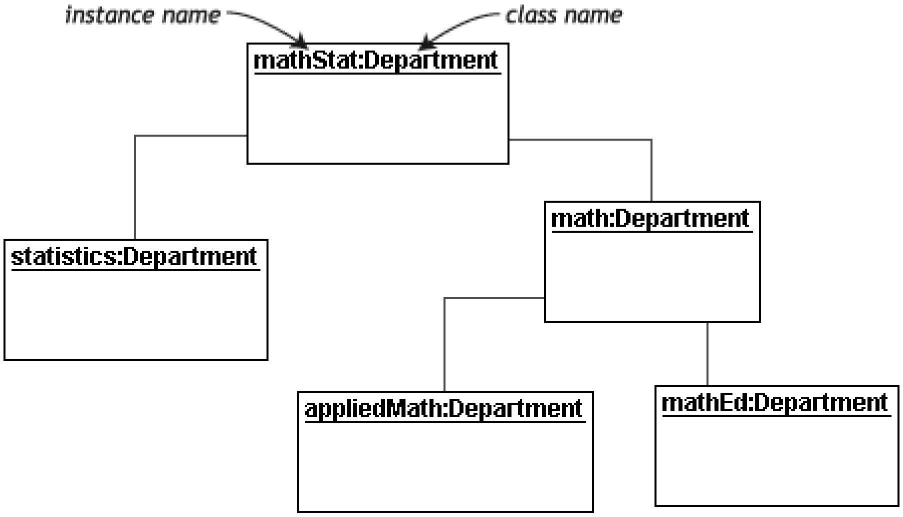
Packages appear as rectangles with small tabs at the top. The package name is on the tab or inside the rectangle. The dotted arrows are **dependencies**. One package depends on another if changes in the other could possibly force changes in the first.

Object diagrams show instances instead of classes. They are useful for explaining small pieces with complicated relationships, especially recursive relationships.

This small class diagram shows that a university **Department** can contain lots of other **Departments**.



The object diagram below instantiates the class diagram, replacing it by a concrete example.



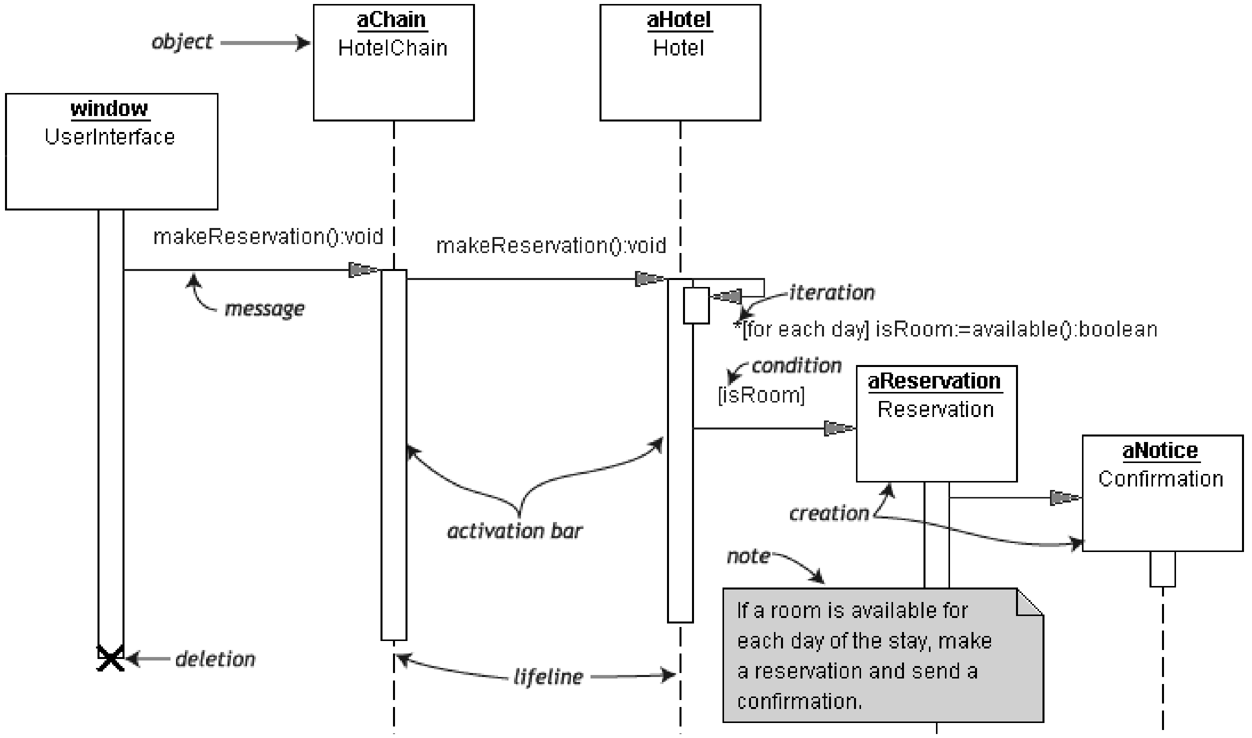
Each rectangle in the object diagram corresponds to a single instance. Instance names are underlined in UML diagrams. Class or instance names may be omitted from object diagrams as long as the diagram meaning is still clear.

6.4 Sequence Diagrams

Class and object diagrams are static model views. **Interaction diagrams** are dynamic. They describe how objects collaborate.

A **sequence diagram** is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is **a Reservation window**.



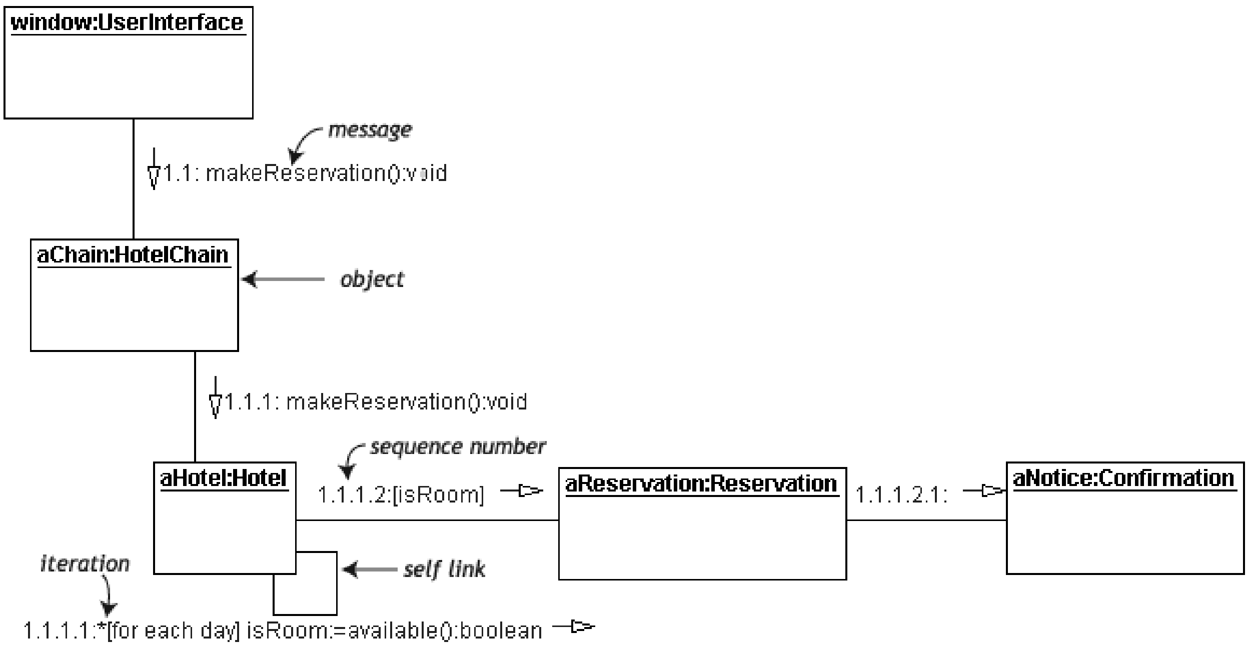
The **Reservation window** sends a makeReservation() message to a **HotelChain**. The **HotelChain** then sends a makeReservation() message to a **Hotel**. If the **Hotel** has available rooms, then it makes a **Reservation** and a **Confirmation**.

Each vertical dotted line is a **lifeline**, representing the time that an object exists. Each arrow is a message call. An arrow goes from the sender to the top of the **activation bar** of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

In our diagram, the **Hotel** issues a **self call** to determine if a room is available. If so, then the **Hotel** creates a **Reservation** and a **Confirmation**. The asterisk on the self call means **iteration** (to make sure there is available room for each day of the stay in the hotel). The expression in square brackets, [ ], is a **condition**. The diagram has a clarifying **note**, which is text inside a dog-eared rectangle. Notes can be put into any kind of UML diagram.

6.5 Collaboration Diagrams

Collaboration diagrams are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links.

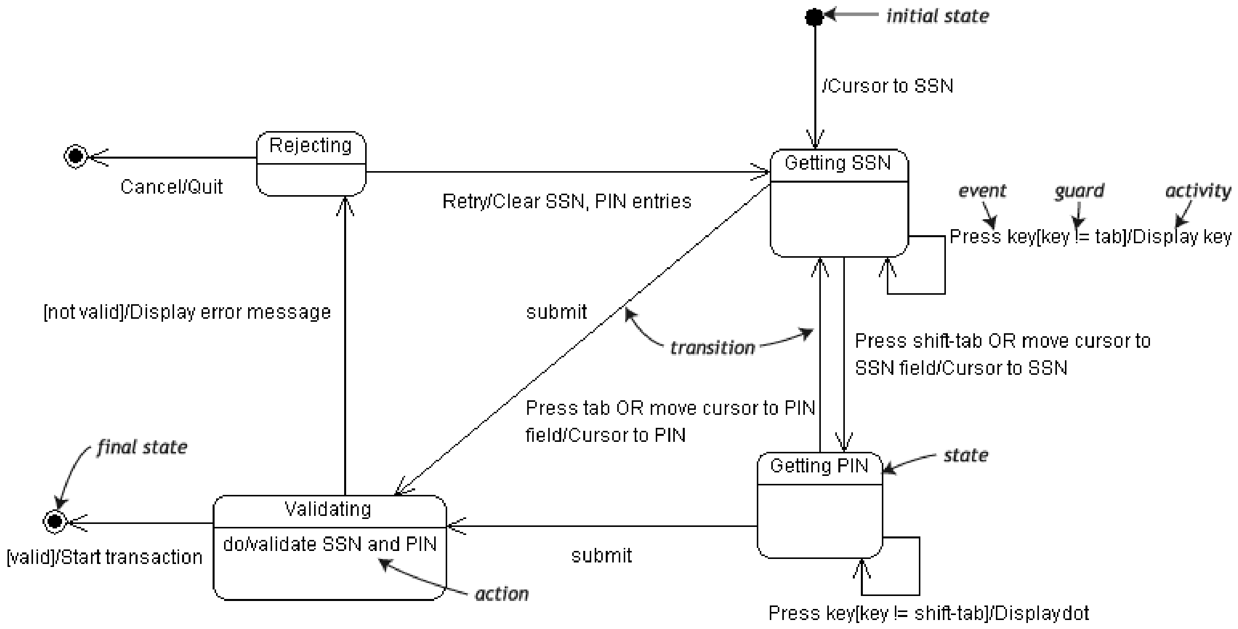


The object-role rectangles are labeled with either class or object names (or both). Class names are preceded by colons (:).

Each message in a collaboration diagram has a **sequence number**. The top-level message is numbered 1. Messages at the same level (sent during the same call) have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

6.6 State Chart Diagrams

Objects have behaviors and state. The state of an object depends on its current activity or condition. A **state chart diagram** shows the possible states of the object and the transitions that cause a change in state. Our example diagram models the login part of an online banking system. Logging in consists of entering a valid social security number and personal id number, then submitting the information for validation. Logging in can be factored into four non-overlapping states: **Getting SSN**, **Getting PIN**, **Validating**, and **Rejecting**. From each state comes a complete set of **transitions** that determine the subsequent state.


initial state
/Cursor to SSN
Rejecting
Cancel/Quit
Retry/Clear SSN, PIN entries
Getting SSN
event
guard
activity
Press key[key != tab]/Display key
[not valid]/Display error message
submit
transition
Press shift-tab OR move cursor to SSN field/Cursor to SSN
Press tab OR move cursor to PIN field/Cursor to PIN
final state
Getting PIN
state
Validating
do/validate SSN and PIN
[valid]/Start transaction
submit
action
Press key[key != shift-tab]/Displaydot

States are rounded rectangles. Transitions are arrows from one state to another. Events or conditions that trigger transitions are written beside the arrows. Our diagram has two self-transition, one on **Getting SSN** and another on **Getting PIN**.

The initial state (black circle) is a dummy to start the action. Final states are also dummy states that terminate the action.

The action that occurs as a result of an event or condition is expressed in the form /action. While in its **Validating** state, the object does not wait for an outside event to trigger a transition. Instead, it performs an activity. The result of that activity determines its subsequent state.

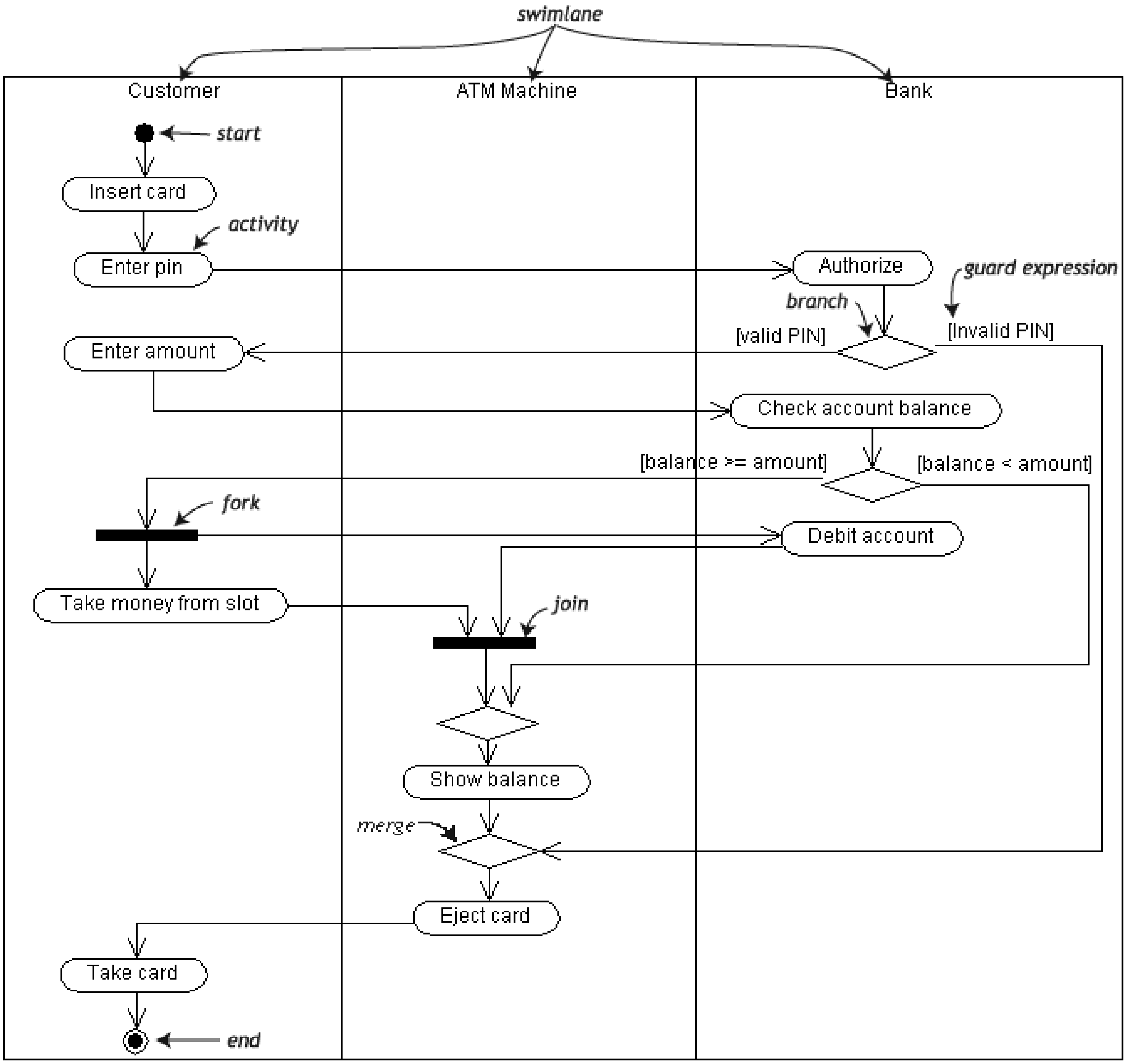
6.7 Activity Diagrams

An **activity diagram** is essentially a fancy flowchart. Activity diagrams and statechart diagrams are related. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one another.

For our example, we used the following process.

"Withdraw money from a bank account through an ATM."

The three involved classes (people, etc.) of the activity are **Customer**, **ATM**, and **Bank**. The process begins at the black start circle at the top and ends at the concentric white/black stop circles at the bottom. The activities are rounded rectangles.



Activity diagrams can be divided into object **swim-lanes** that determine which object is responsible for which activity. A single **transition** comes out of each activity, connecting it to the next activity.

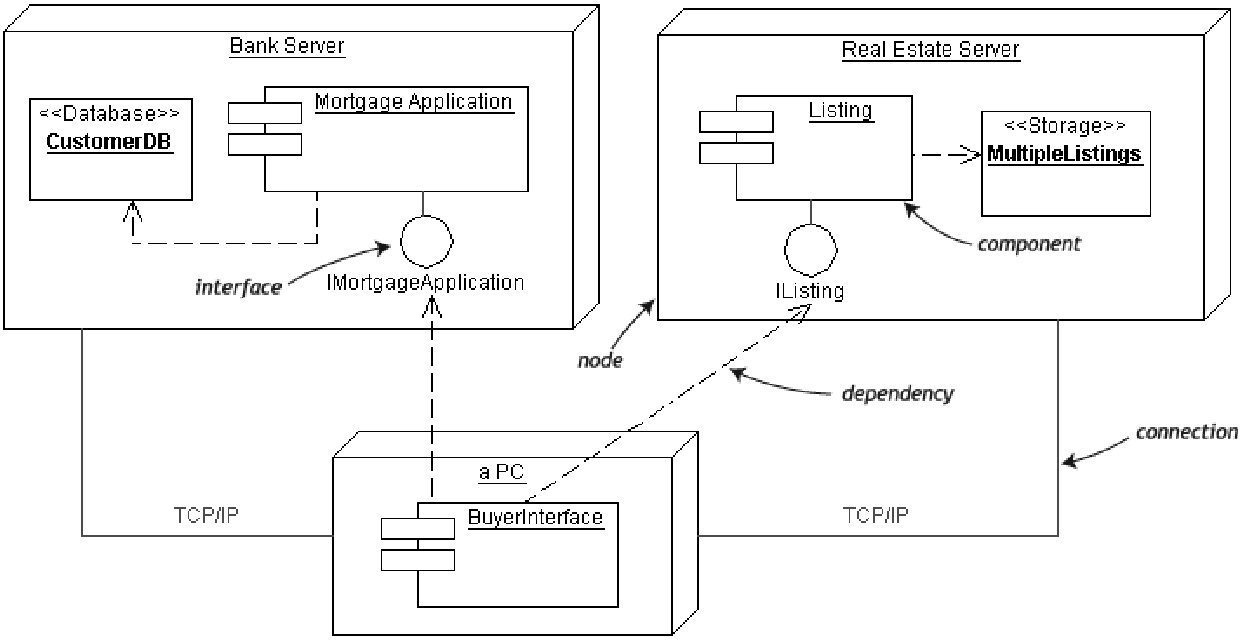
A transition may **branch** into two or more mutually exclusive transitions. **Guard expressions** (inside [ ]) label the transitions coming out of a branch. A branch and its subsequent **merge** marking the end of the branch appear in the diagram as hollow diamonds.

A transition may **fork** into two or more parallel activities. The fork and the subsequent **join** of the threads coming out of the fork appear in the diagram as solid bars.

6.8 Component and Deployment Diagrams

A **component** is a code module. Component diagrams are physical analogs of class diagram. **Deployment diagrams** show the physical configurations of software and hardware.

The following deployment diagram shows the relationships among software and hardware components involved in real estate transactions.



The physical hardware is made up of **nodes**. Each component belongs on a node. Components are shown as rectangles with two tabs at the upper left.

CHAPTER 7:

Software Testing

7.1 Definition and Life Cycle

Definition

Software testing is a process of executing a program with the intent of finding error. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should. It is oriented towards “detection”.

Software testing should not be confused with software quality assurance. Software quality assurance is a process of setting up quality standards and benchmarks to ensure the achievement of expected quality where as software testing is a process of comparing the quality achieved with quality expected. It is focused on minimizing this gap and ensures the delivering of a high quality product. The difference can be more easily understood by the example of a chef who follows a recipe to cook a particular dish. Recipe suggests the various contents, ratio of the contents, level of heat, process of cooking etc. to make sure that the intended dish must taste at its best so a recipe is nothing but the ***quality assurance*** guide for the dish. Once the dish is cooked, chef tastes it himself before serving it to the guests to make sure it tastes the same as expected. So chef is actually ***testing*** the dish.

Why is testing done

Software testing is done for the below mentioned reasons:

- Need to ensure system works before it is turned over to the business.
- Error Free Superior Product.
- Quality Assurance to the Client.
- Competitive advantage.
- Cut down Costs.

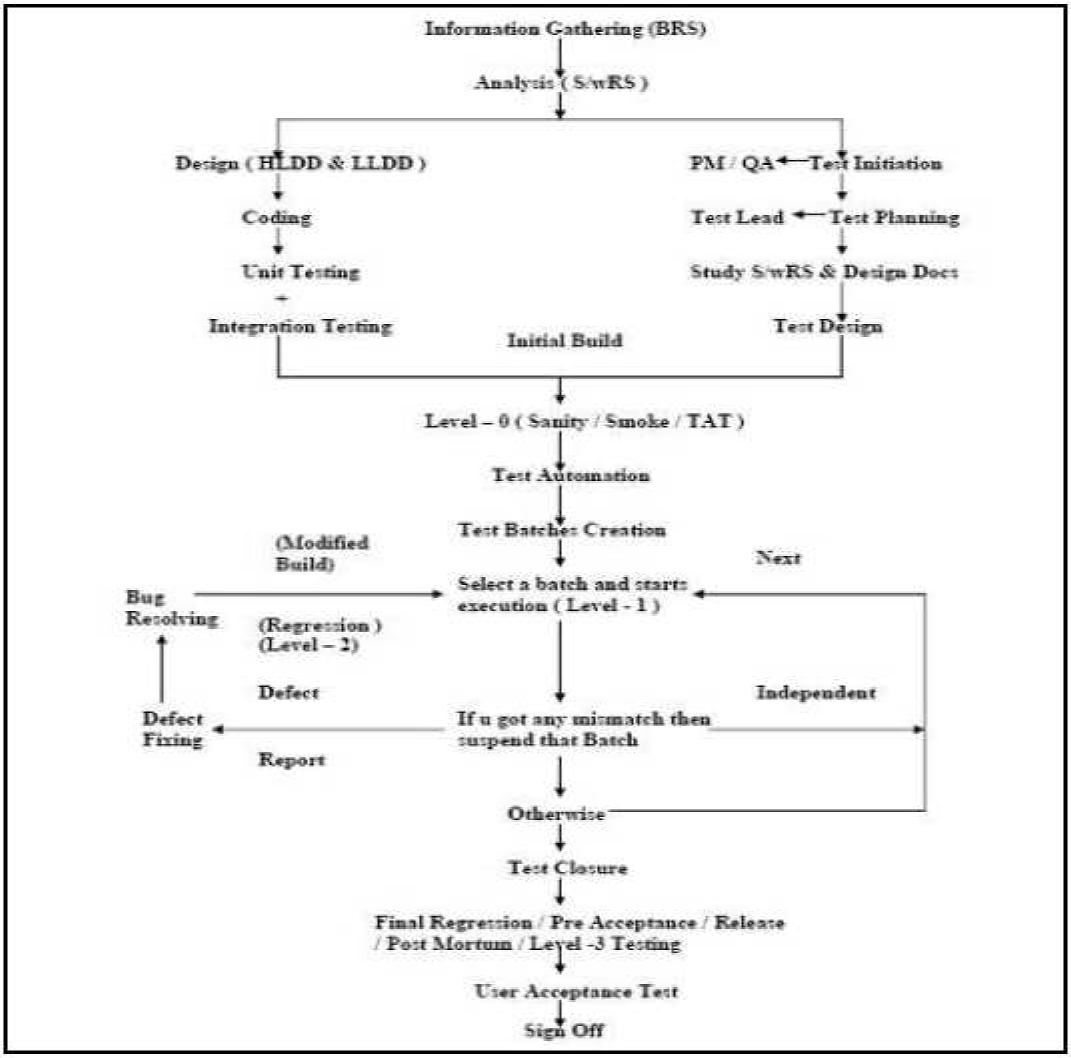
Software Testing Life Cycle

Once software is developed, the sequence of activities which take place in QA department is as below:

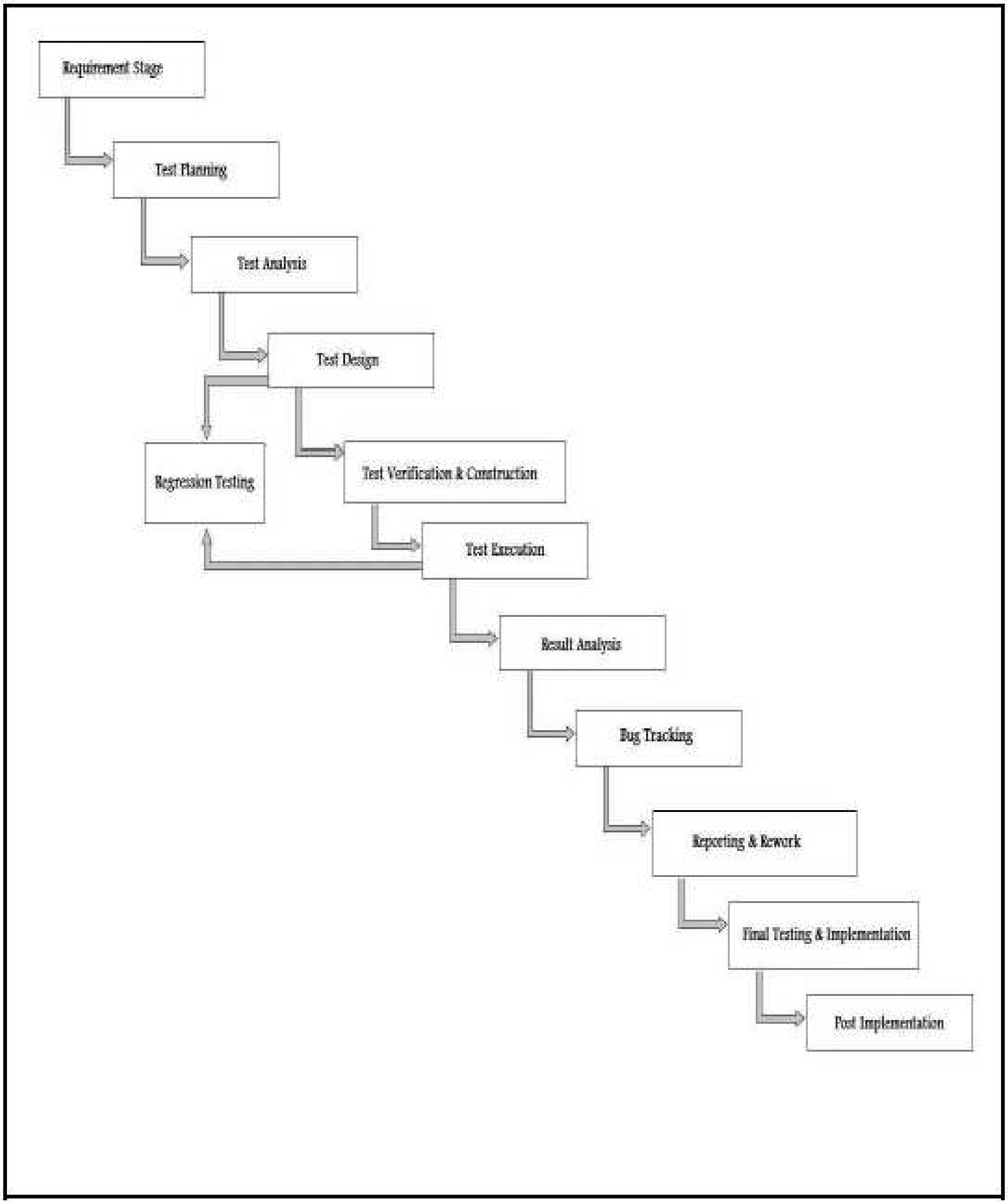
- Understand business Requirements
- Understand the Application
- Prepare the Test Plan
- Prepare Test Condition
- Review Test Plan and Test Condition
- Prepare Test Case

- Review Test Case
- Prepare Test Data
- Prepare Test Script
- Execute Test Case
- Report the Defect
- Retest the Defect
- Attend Project Meeting calls, defect calls and Requirement calls
- Prepare the Traceability Matrix
- Send Daily report or weekly report to Manager

One end to end software testing process, depending on the size of the application, can be small or a lengthy one. A typical end-to-end software testing process can be explained with the help of the following diagram:



The sequence of phases involved in software testing life cycle is depicted in the diagram below:



1. Requirement Stage

This is the initial stage of the life cycle process in which the developers take part in analyzing the requirements for designing a product. Testers can also involve themselves as they can think from the users' point of view which the developers may not. Thus a panel of developers, testers and users can be formed. Formal meetings of the panel can be held in order to document the requirements discussed which can be further used as software requirements specifications or SRS.

2. Test Planning

Test planning is predetermining a plan well in advance to reduce further risks. Without a good plan, no work can lead to success be it software-related or routine work. A test plan document plays an important role in achieving a process-oriented approach. Once the requirements of the project are confirmed, a test plan is documented. The test plan structure is as follows:

- **Introduction:** This describes the objective of the test plan.
- **Test Items:** The items that are referred to prepare this document will be listed here such as SRS, project plan.
- **Features to be tested:** This describes the coverage area of the test plan, ie. the list of features that are to be tested that are based on the implicit and explicit requirements from the customer.
- **Features not to be tested:** The incorporated or comprised features that can be skipped from the testing phase are listed here. Features that are out of scope of testing, like incomplete modules or those on low severity eg. GUI features that don't hamper the further process can be included in the list.
- **Approach:** This is the test strategy that should be appropriate to the level of the plan. It should be in acceptance with the higher and lower levels of the plan.
- **Item pass/fail criteria:** Related to the show stopper issue. The criterion which is used has to explain which test item has passed or failed.
- **Suspension criteria and resumption requirements:** The suspension criterion specifies the criterion that is to be used to suspend all or a portion of the testing activities, whereas resumption criterion specifies when testing can resume with the suspended portion.
- **Test deliverable:** This includes a list of documents, reports, charts that are required to be presented to the stakeholders on a regular basis during testing and when testing is completed.
- **Testing tasks:** This stage is needed to avoid confusion whether the defects should be reported for future function. This also helps users and testers to avoid incomplete functions and prevent waste of resources.
- **Environmental needs:** The special requirements of that test plan depending on the environment in which that application has to be designed are listed here.
- **Responsibilities:** This phase assigns responsibilities to the person who can be held responsible in case of a risk.
- **Staffing and training needs:** Training on the application/system and training on the testing tools to be used needs to be given to the staff members who are responsible for the application.
- **Risks and contingencies:** This emphasizes on the probable risks and various events that can occur and what can be done in such situation.
- **Approval:** This decides who can approve the process as complete and allow the project to proceed to the next level that depends on the level of the plan.

3. Test Analysis

Once the test plan documentation is done, the next stage is to analyze what types of software testing should be carried out at the various stages of SDLC.

4. Test Design

Test design is done based on the requirements of the project documented in the SRS. This phase decides whether manual or automated testing is to be done. In automation testing, different paths for testing are to be identified first and writing of scripts has to be done if required. There originates a need for an end to end checklist that covers all the features of the project.

5. Test Verification and Construction

In this phase test plans, the test design and automated script tests are completed. Stress and performance testing plans are also completed at this stage. When the development team is done with a unit of code, the testing team is required to help them in testing that unit and reporting of the bug if found. Integration testing and bug reporting is done in this phase of the software testing life cycle.

6. Test Execution

Planning and execution of various test cases is done in this phase. Once the unit testing is completed, the functionality of the tests is done in this phase. At first, top level testing is done to find out top level failures and bugs are reported immediately to the development team to get the required workaround. Test reports have to be documented properly and the bugs have to be reported to the development team.

7. Result Analysis

Once the bug is fixed by the development team, i.e after the successful execution of the test case, the testing team has to retest it to compare the expected values with the actual values, and declare the result as pass/fail.

8. Bug Tracking

This is one of the important stages as the Defect Profile Document (DPD) has to be updated for letting the developers know about the defect. Defect Profile Document contains the following

- **Defect Id:** Unique identification of the Defect.
- **Test Case Id**: Test case identification for that defect.
- **Description:** Detailed description of the bug.
- **Summary:** This field contains some keyword information about the bug, which can help in minimizing the number of records to be searched.
- **Defect Submitted By:** Name of the tester who detected/reported the bug.
- **Date of Submission:** Date at which the bug was detected and reported.

- **Build No.:** Number of test runs required.
- **Version No.:** The version information of the software application in which the bug was detected and fixed.
- **Assigned To:** Name of the developer who is supposed to fix the bug.
- **Severity:** Degree of severity of the defect.
- **Priority:** Priority of fixing the bug.
- **Status:** This field displays current status of the bug.

Reporting and Rework

Testing is an iterative process. The bug once reported and as the development team fixes the bug, it has to undergo the testing process again to assure that the bug found is resolved. Regression testing has to be done. Once the Quality Analyst assures that the product is ready, the software is released for production. Before release, the software has to undergo one more round of top level testing. Thus testing is an ongoing process.

Final Testing and Implementation

This phase focuses on the remaining levels of testing, such as acceptance, load, stress, performance and recovery testing. The application needs to be verified under specified conditions with respect to the SRS. Various documents are updated and different matrices for testing are completed at this stage of the software testing life cycle.

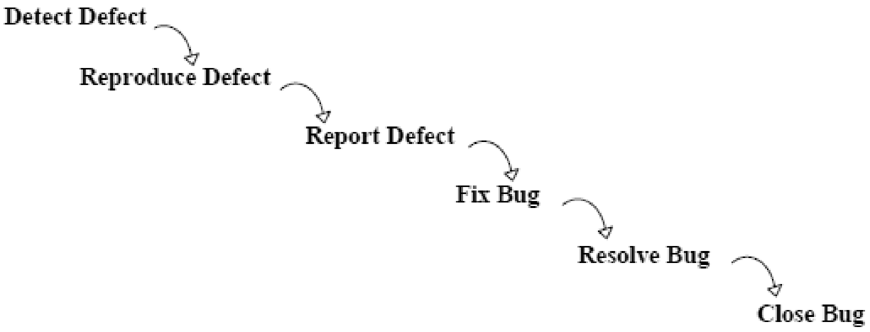
Post Implementation

Once the tests are evaluated, the recording of errors that occurred during various levels of the software testing life cycle is done. Creating plans for improvement and enhancement is an ongoing process. This helps to prevent similar problems from occurring in the future projects. In short, planning for improvement of the testing process for future applications is done in this phase.

7.1.1 Software Defect Cycle

During the software testing, software testers try to find the bugs or defects in the software. Defect can be defined as “Deviation of the requirement”. No software is worthwhile as long as it has defects. All the defects found during the testing are logged in a defect log and are assigned severity level (High, Medium or Low) and defect status (New, Open, Fixed, Closed, Reopen etc.) and reported to the test lead and development team.

A typical defect life cycle can be depicted as the following diagram:



Bug status description:

These are various stages of bug life cycle. The status caption may vary depending on the bug tracking system you are using.

1) New: When QA logs a new bug.

2) Deferred: If the bug is not related to current build or can not be fixed in this release or bug is not important to fix immediately then the project manager can set the bug status as deferred.

3) Assigned: ‘Assigned to’ field is set by project lead or manager and assigns bug to developer.

4) Resolved/Fixed: When developer makes necessary code changes and verifies the changes then he/she can make bug status as ‘Fixed’ and the bug is passed to testing team.

5) Could not reproduce: If developer is not able to reproduce the bug by the steps given in bug report by QA then developer can mark the bug as ‘CNR’. QA needs action to check if bug is reproduced and can assign to developer with detailed reproducing steps.

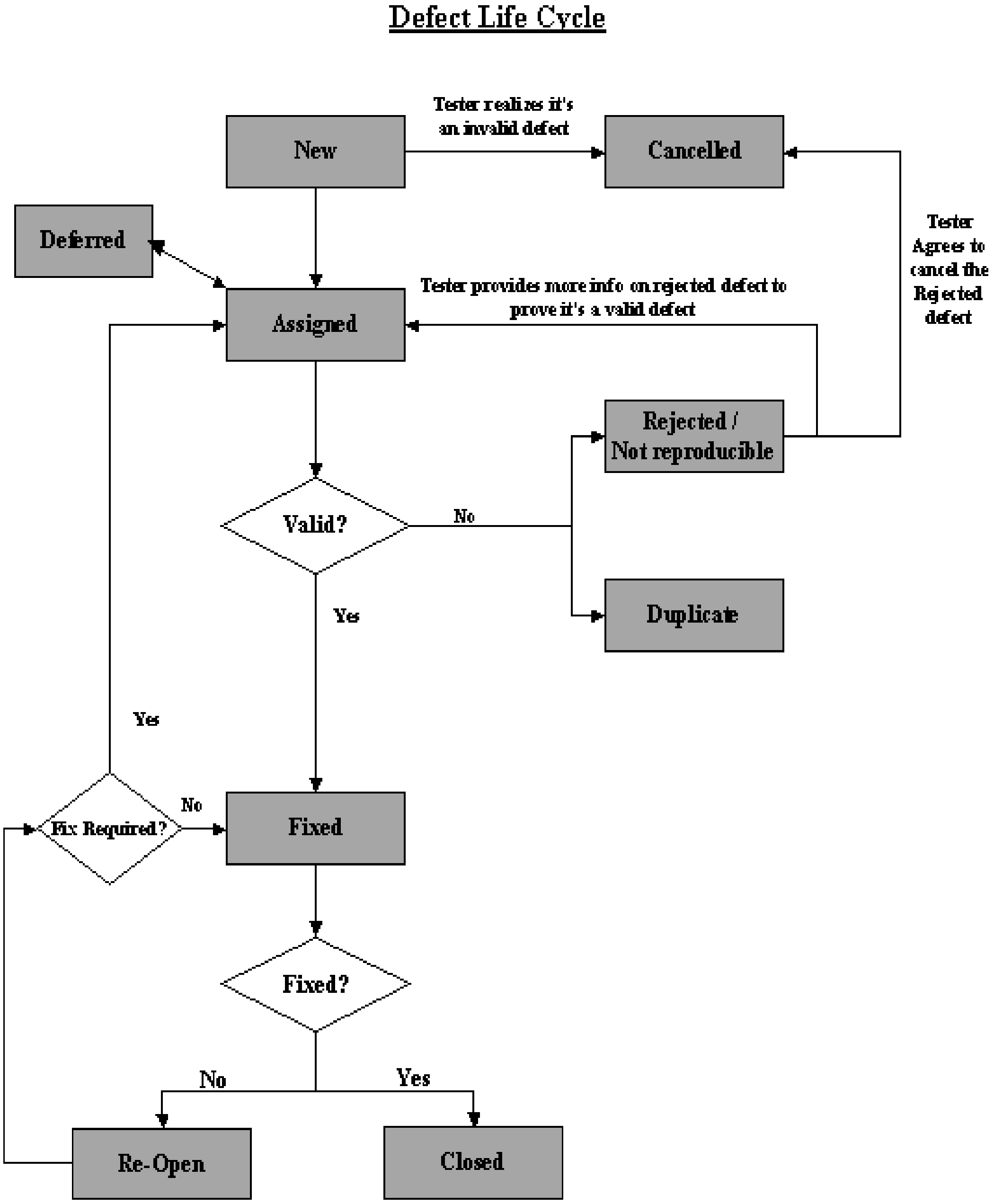
6) Need more information: If developer is not clear about the bug reproduce steps provided by QA to reproduce the bug, then he/she can mark it as “Need more information’. In this case QA needs to add detailed reproducing steps and assign bug back to dev for fix.

7) Reopen: If QA is not satisfy with the fix and if bug is still reproducible even after fix then QA can mark it as ‘Reopen’ so that developer can take appropriate action.

8) Closed: If bug is verified by the QA team and if the fix is ok and problem is solved then QA can mark bug as ‘Closed’.

9) Rejected/Invalid: Some times developer or team lead can mark the bug as Rejected or invalid if the system is working according to specifications and bug is just due to some misinterpretation.

Defect Status Cycle can be explained by the following diagram:


Defect Life Cycle
New
Tester realizes it's an invalid defect
Cancelled
Deferred
Tester Agrees to cancel the Rejected defect
Tester provides more info on rejected defect to prove it's a valid defect
Assigned
Rejected / Not reproducible
Valid?
No
Duplicate
Yes
Yes
No
Fix Required?
Fixed
Fixed?
No
Yes
Re-Open
Closed

7.1.2 Types of Software Testing

Software testing can be performed either manually, with other automated software or with a combination of both. The types of software testing and their explanations are as below:

User Acceptance Testing	Testing the system with the intent of confirming readiness of the product and customer acceptance.
Ad Hoc Testing	Testing without a formal test plan or outside of a test plan. With some projects this type of testing is carried out as an adjunct to formal testing. If carried out by a skilled tester, it can often find problems that are not caught in regular testing. Sometimes, if testing occurs very late in the development cycle, this will be the only kind of testing that can be performed. Sometimes ad hoc testing is referred to as exploratory testing.
Alpha Testing	Testing after code is mostly complete or contains most of the functionality and prior to users being involved. Sometimes a select group of users are involved. More often this testing will be performed in-house or by an outside testing firm in close cooperation with the software engineering department.
Automated Testing	Software testing that utilizes a variety of tools to automate the testing process and when the importance of having a person manually testing is diminished. Automated testing still requires a skilled quality assurance professional with knowledge of the automation tool and the software being tested to set up the tests.
Beta Testing	Testing after the product code is complete. Betas are often widely distributed or even distributed to the public at large in hopes that they will buy the final product when it is released.
Black Box Testing	Testing software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as a specification

	or requirements document.
Compatibility Testing	Testing used to determine whether other system software components such as browsers, utilities, and competing software will conflict with the software being tested.
Configuration Testing	Testing to determine how well the product works with a broad range of hardware/peripheral equipment configurations as well as on different operating systems and software.
Functional Testing	Testing two or more modules together with the intent of finding defects, demonstrating that defects are not present, verifying that the module performs its intended functions as stated in the specification and establishing confidence that a program does what it is supposed to do.
Independent Verification and Validation (IV&V)	The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The individual or group doing this work is not part of the group or organization that developed the software. A term often applied to government work or where the government regulates the products, as in medical devices.
Installation Testing	Testing with the intent of determining if the product will install on a variety of platforms and how easily it installs.
Integration Testing	Testing two or more modules or functions together with the intent of finding interface defects between the modules or functions. Testing completed at as a part of unit or functional testing, and sometimes, becomes its own standalone test phase. On a larger level, integration testing can involve a putting together of groups of modules and functions with the goal of completing and verifying that the system meets the system requirements. (see system testing)

Load Testing

Testing with the intent of determining how well the product handles competition for system resources. The competition may come in the form of network traffic, CPU utilization or memory allocation.

Performance Testing

Testing with the intent of determining how quickly a product handles a variety of events. Automated test tools geared specifically to test and fine-tune performance are used most often for this type of testing.

Pilot Testing

Testing that involves the users just before actual release to ensure that users become familiar with the release contents and ultimately accept it. Often is considered a Move-to-Production activity for ERP releases or a beta test for commercial products. Typically involves many users, is conducted over a short period of time and is tightly controlled. (see beta testing)

Regression Testing

Testing with the intent of determining if bug fixes have been successful and have not created any new problems. Also, this type of testing is done to ensure that no degradation of baseline functionality has occurred.

Security Testing

Testing of database and network software in order to keep company data and resources secure from mistaken/accidental users, hackers, and other malevolent attackers.

Software Testing

The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The organization and management of individuals or groups doing this work is not relevant. This term is often applied to commercial products such as internet applications. (contrast with independent verification and validation)

Stress Testing

Testing with the intent of determining how well a product performs when a load is placed on the system resources that nears and then exceeds capacity.

System Integration Testing	Testing a specific hardware/software installation. This is typically performed on a COTS (commerical off the shelf) system or any other system comprised of disparent parts where custom configurations and/or unique installations are the norm.
User Acceptance Testing	See Acceptance Testing.
White Box Testing	Testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose.

7.2 Creating Software Test Plan

A software test plan focuses at planning the software testing process and encompasses defining the in-scope, out of scope, objective, test strategy, test phases, test schedule, resources required, roles and responsibilities of the team members involved, error measurement system, reporting requirements, communication channel, sign off process etc. It aims at complying with the "conformance to the requirements" with a goal of establishing a list of tasks, if performed, will identify all the requirements that have not been met.

The test plan represents the overall approach to the test. In many ways, the test plan serves as a summary of the test activities that will be performed. It shows how the tests will be organized, and outlines all of the testers' needs that must be met in order to properly carry out the test.

A test plan usually contains the following information:

1. Introduction

Covers the Introduction of the project, background information and objective of the test initiative.

2. Scope

Covers the scope of the test in terms of areas to be covered such as

3. Developing Software Test Strategy

3.1. System Test
3.2. Performance Test
3.3. Security Test
3.4. Automated Test
3.5. Stress and Volume Test
3.6. Recovery Test

3.7. Documentation Test
3.8. Beta Test
3.9. User Acceptance Test

4. Environment Requirements
4.1. Data Entry workstations
4.2 Main Frame

5. Test Schedule
6. Control Procedures
6.1 Reviews
6.2 Bug Review meetings
6.3 Change Request
6.4 Defect Reporting

7. Functions To Be Tested
8. Resources and Responsibilities
8.1. Resources
8.2. Responsibilities
9. Deliverables
10. Suspension / Exit Criteria
11. Resumption Criteria
12. Dependencies
12.1 Personnel Dependencies
12.2 Software Dependencies
12.3 Hardware Dependencies
12.3 Test Data & Database
13. Risks
13.1. Schedule
13.2. Technical
13.3. Management
13.4. Personnel
13.5 Requirements
14. Tools
15. Documentation
16. Approvals

7.3 Software Test Scenario, Test Cases & Test Conditions

Test Scenario:

A test scenario is a set of test cases that ensures that the business process flows are tested from end to end. They may be independent tests or a series of tests that follow each other each dependent on the output of the previous one. A test scenario can also be referred as the flow of data from one end to another end in a process. It suggests the possible scenario for which an application can be tested. Usually test cases are derived from test scenarios and test scenarios are derived from use cases.

Test Case:

A test case is – "A set of conditions or variables under which it describes an application's input, action or event and an expected response for a tester to determine whether a software application is working as expected."

Test Case is a commonly used term for a specific test. This is usually the smallest unit of testing and while performing the manual testing sometimes it is also referred as **test script** but in automation testing test script is a program written to test the functionality of an application. It is a set of system readable instructions to automate the testing. It is a set of inputs, execution, preconditions and expected outcomes developed for a particular objective such as to verify compliance with a specific requirement. Writing test cases is necessary for standardization and minimizes the ad-hoc approach in testing.

A test case is usually a single step, or occasionally a sequence of steps, to test the correct behavior/ functionality, features of an application. An expected result or expected outcome is usually given. Additional information that may be included:

- test case ID
- test case description
- test step or order of execution number
- related requirement(s)
- depth
- test category
- author
- check boxes for whether the test can be or has been automated
- pass/fail
- remarks

A test case can be both positive and a negative test case. In order to fully test that all the requirements of an application are met, there must be at least two test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases. Keeping track of the link between the requirement and the test is frequently done using a traceability matrix. Written test cases should include a description of the functionality to be tested, and the preparation required to ensure that the test can be conducted.

<u>Example of Positive Test Case:</u>

Test Case No: *TC 1.1*

Test Case Name: *Verify that user can Login myCRM with correct username and Password*

Project Name: *myCRM*

Module Name: *Login*

Date*:* *XX-XX-XXXX*

Author: *John Schultz*

Pre-Conditions: *User must have Valid URL, Username and Password*

SNo.	Description	Expected result	Pass/fail	comments
1	Enter URL; www.mycrm.com	myCRM Login Page is displayed	PASS	
2	Enter Username: John	John is displayed in username text box	PASS	
3	Enter Password: Schultz001	Password is displayed in password textbox in encrypted format	PASS	
4	Click on Login Button	Systems displays the Home Page	PASS	

<u>Example of Positive Test Case:</u>

Test Case No: *TC 1.1*

Test Case Name: *Verify that user CAN NOT Login myCRM with invalid username and Password*

Project Name: *myCRM*

Module Name: *Login*

Date*:* *XX-XX-XXXX*

Author: *John Schultz*

Pre-Conditions: *None*

SNo.	Description	Expected result	Pass/fail	comments
1	Enter URL; www.mycrm.com	myCRM Login Page is displayed	PASS	
2	Enter Username: John	John is displayed in username text box	PASS	
3	Enter Password: Any invalid password	Password is displayed in password textbox in encrypted format	PASS	
4	Click on Login Button	Systems displays the error message: "Oops! Wrong Password! Please try again"	PASS	

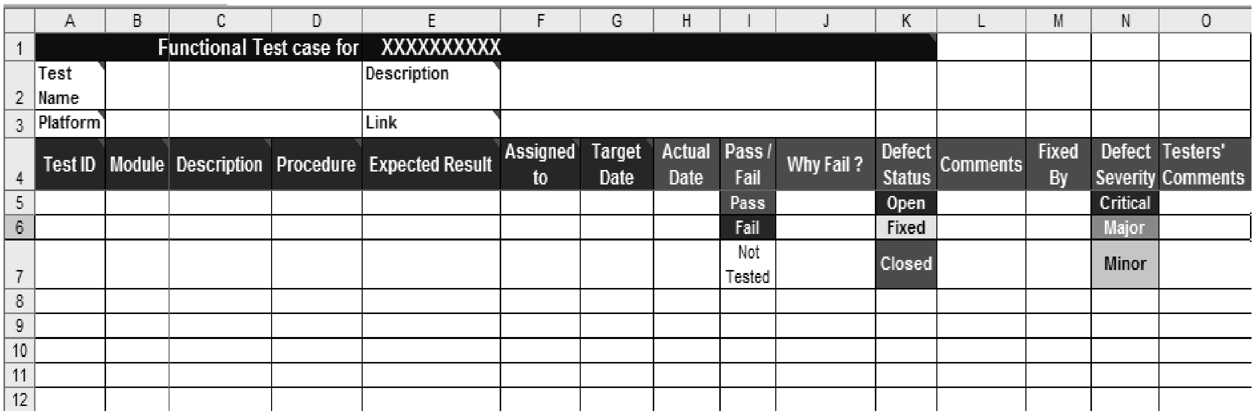
A negative test case can have multiple possible combinations and that is where we should prioritize which combinations are High/Medium or Low priority combinations as not all combinations are necessary to test:

- Wrong Login/ Right Password
- Right Login/ Wrong Password
- Wrong Login/ Wrong Password
- No Login/ Right Password
- No Login/ Wrong Password
- Right Login/ No Password
- No Login/ No Password

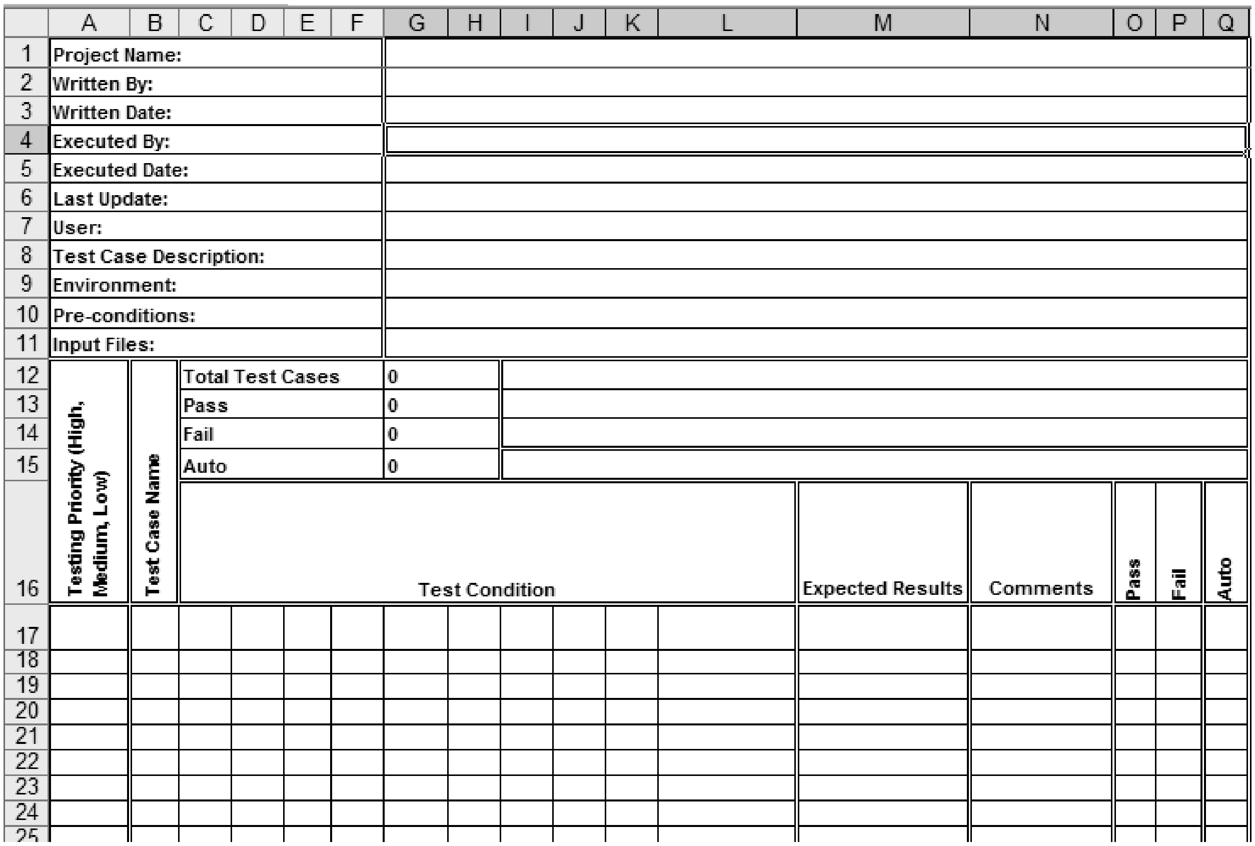
Test Condition:

Test condition is the process that we follow to test an application. System is supposed to fulfill one condition in order to move to the next step e.g. If a user intends to check his current health insurance invoice for the total pending amount to be paid, user has to login in the system and essential condition will be system will let the user access his account only when it will validate the user name and password. Test scenario and test condition are used highly interchangeably.

There are different ways of writing test cases varying company to company but here are some variations of test case formats:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1			Functional Test case for		XXXXXXXXXX										
2	Test Name				Description										
3	Platform				Link										
4	Test ID	Module	Description	Procedure	Expected Result	Assigned to	Target Date	Actual Date	Pass / Fail	Why Fail ?	Defect Status	Comments	Fixed By	Defect Severity	Testers' Comments
5									Pass		Open			Critical	
6									Fail		Fixed			Major	
7									Not Tested		Closed			Minor	
8															
9															
10															
11															
12															

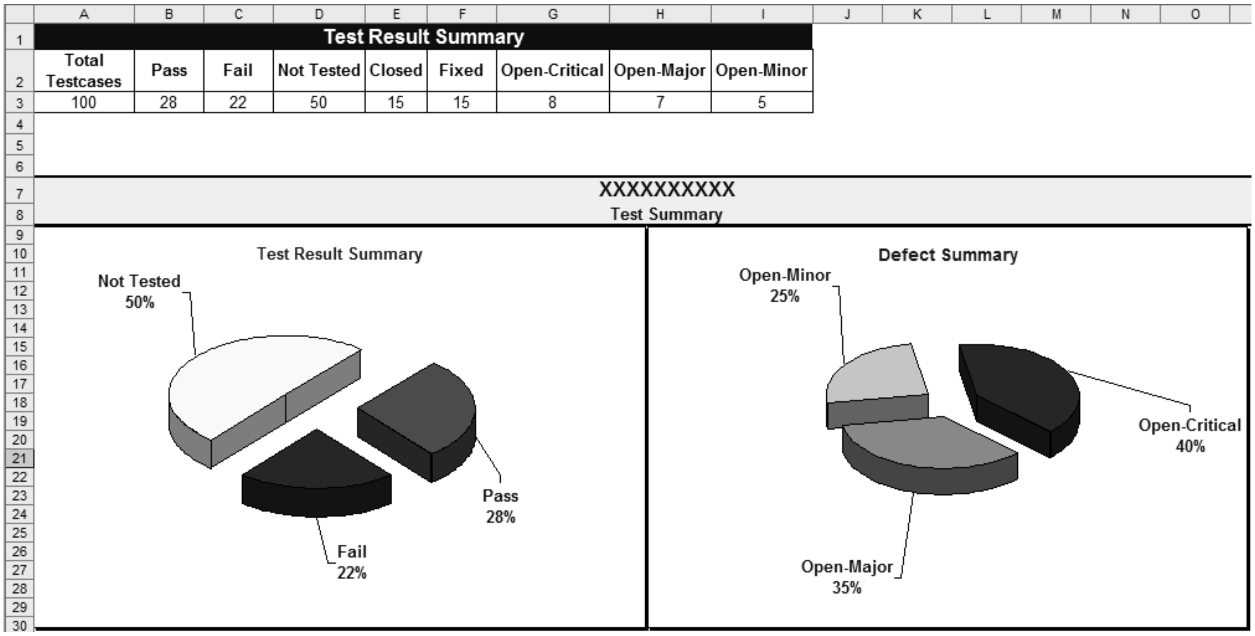


Project Name:	
Written By:	
Written Date:	
Executed By:	
Executed Date:	
Last Update:	
User:	
Test Case Description:	
Environment:	
Pre-conditions:	
Input Files:	

Total Test Cases	0
Pass	0
Fail	0
Auto	0

Testing Priority (High, Medium, Low)	Test Case Name	Test Condition	Expected Results	Comments	Pass	Fail	Auto

And finally the test results are analyzed by the test team lead. Test results looks something like this:



Test Result Summary

Total Testcases	Pass	Fail	Not Tested	Closed	Fixed	Open-Critical	Open-Major	Open-Minor
100	28	22	50	15	15	8	7	5

XXXXXXXXXX

Test Summary

SAMPLE PROJECT DOCUMENTS

AND

TEMPLATES

Project Scope Document

Project Name	
Document Version	
Prepared by	
Date created	
Approved by	
Date approved	

Revision History

Name	Date	Reason For Changes	Version

1. Business Requirements

<The business requirements provide the foundation and reference for all detailed requirements development. You may gather business requirements from the customer or development organization’s senior management, an executive sponsor, a project visionary, product management, the marketing department, or other individuals who have a clear sense of why the project is being undertaken and the ultimate value it will provide, both to the business and to customers.>

1.1. Background

<This section summarizes the rationale for the new product. Provide a general description of the history or situation that leads to the recognition that this product should be built.>

1.2. Business Opportunity

<Describe the market opportunity that exists or the business problem that is being solved. Describe the market in which a commercial product will be competing or the environment in which an information system will be used. This may include a brief comparative evaluation of existing products and potential solutions, indicating why the proposed product is attractive. Identify the problems that cannot currently be solved without the product, and how the product fits in with market trends or corporate strategic directions.>

1.3. Business Objectives and Success Criteria

<Describe the important business objectives of the product in a way that is quantitative and measurable. The value provided to customers is described in section 1.4, so this section should focus on the value provided to the business. This could include estimates of revenue or cost savings, return on investment analysis, or target release dates. Determine how success will be defined and measured on this project, and describe the factors that are likely to have the greatest impact on achieving that success. Include things within the direct control of the organization, as well as external factors. Establish measurable criteria to assess whether the business objectives have been met.>

1.4. Customer or Market Needs

<Describe the needs of typical customers or market segments, including needs that are not yet met by the marketplace or by existing systems. You may wish to describe problems customers currently encounter that the new product will (or will not) address and how the product would be used by customers. Identify the customer hardware and software environment in which the product must operate. Define at a high level any known critical interface or performance requirements. Avoid including any design or implementation details. Present the requirements in a numbered list so that more detailed user or functional requirements can be traced to them.>

1.5. Business Risks

<Summarize the major business risks associated with developing this product, such as marketplace competition, timing issues, user acceptance, implementation issues, or possible negative impacts on the business. Estimate the severity of the risks and identify any risk mitigation actions that could be taken.>

2. Vision of the Solution

<This section establishes a long-term vision for the system to be built to address the business objectives. This vision will provide the context for making decisions throughout the course of the product development life cycle. The vision should not include detailed functional requirements or project planning information.>

2.1. Vision Statement

<Write a concise vision statement that summarizes the purpose and intent of the new product and describes what the world will be like when it includes the product. The vision statement should reflect a balanced view that will satisfy the needs of diverse customers as well as those of the developing organization. It may be somewhat idealistic, but it should be grounded in the realities of existing or anticipated customer markets, enterprise architectures, organizational strategic directions, and cost and resource limitations.>

2.2. Major Features

<Include a numbered list of the major features of the new product, emphasizing those features that distinguish it from previous or competing products. Specific user requirements and functional requirements may be traced back to these features.>

2.3. Assumptions and Dependencies

<Record any assumptions that were made when conceiving the project and writing this vision and scope document. Note any major dependencies the project must rely upon for success, such as specific technologies, third-party vendors, development partners, or other business relationships.>

3. Scope and Limitations

<The project scope defines the concept and range of the proposed solution. It’s also important to define what will not be included in the product. Clarifying the scope and limitations helps to establish realistic expectations of the many stakeholders. It also provides a reference frame against which proposed features and requirements changes can be evaluated. Proposed requirements that are out of scope for the envisioned product must be rejected, unless they are so beneficial that the scope should be enlarged to accommodate them (with accompanying changes in budget, schedule, and/or resources).>

3.1. Scope of Initial Release

<Describe the intended major features that will be included in the initial release of the product. Consider the benefits the product is intended to bring to the various customer communities, and generally describe the product features and quality characteristics that will enable it to provide those benefits. Avoid the temptation to include every possible feature that any potential customer category might conceivably want some day. Focus on those features and product characteristics that will provide the most value, at the most acceptable development cost, to the broadest community.>

3.2. Scope of Subsequent Releases

<If a staged evolution of the product is envisioned over time, indicate which major features will be deferred to later releases.>

3.3. Limitations and Exclusions

<Identify any product features or characteristics that a stakeholder might anticipate, but which are not planned to be included in the new product.>

4. Business Context

<This section summarizes some of the business issues around the project, including profiles of major customer categories, assumptions that went into the project concept, and the management priorities for the project.>

4.1. Stakeholder Profiles

<Stakeholders are individuals, groups, or organizations that are actively involved in a project, are affected by its outcome, or can influence its outcome. The stakeholder profiles identify the customers for this product and other stakeholders, and states their major interests in the product. Characterize business-level customers, target market segments, and different user classes, to reduce the likelihood of unexpected requirements surfacing later that cannot be accommodated because of schedule or scope constraints. For each stakeholder category, the profile includes the major value or benefits they will receive from the product, their likely attitudes toward the product, major features and characteristics of interest, and any known constraints that must be accommodated. Examples of stakeholder value include:

- improved productivity
- reduced rework
- cost savings
- streamlined business processes
- automation of previously manual tasks
- ability to perform entirely new tasks or functions
- conformance to current standards or regulations
- improved usability or reduced frustration level compared to current applications

Example

Stakeholder	Major Value	Attitudes	Major Interests	Constraints
executives	increased revenue	see product as avenue to 25% increase in market share	richer feature set than competitors; time to market	maximum budget = $1.4M
editors	fewer errors in work	highly receptive, but expect high usability	automatic error correction; ease of use; high reliability	must run on low-end workstations
legal aides	quick access to data	resistant unless product is keystroke-compatible with current system	ability to handle much larger database than current system; easy to learn	no budget for retraining

4.2. Project Priorities

<Describe the priorities among the project's requirements, schedule, and budget. The table below may be helpful in identifying the parameters around the project's key drivers (top priority objectives), constraints to work within, and dimensions that can be balanced against each other to achieve the drivers within the known constraints.>

Dimension	Driver (state objective)	Constraint (state limits)	Degree of Freedom (state allowable range)
Schedule	release 1.0 to be available by 10/1, release 1.1 by 12/1		
Features			70-80% of high priority features must be included in release 1.0
Quality			90-95% of user acceptance tests must pass for release 1.0, 95-98% for release 1.1
Staff		maximum team size is 6 developers + 4 testers	
Cost			budget overrun up to 15% acceptable without executive review

4.3. Operating Environment

<Describe the environment in which the system will be used and define the major availability, reliability, performance, and integrity requirements. This information will significantly influence the definition of the system's architecture. Consider questions such as:

- Are the users widely distributed geographically or located close to each other? How many time zones are they in?
- When do the users in various locations need to access the system?
- Where is the data generated and used? How far apart are these locations? Does the data from multiple locations need to be combined?
- Are specific maximum response times known for accessing data that might be stored remotely?
- Can the users tolerate service interruptions or is continuous access to the system critical for the operation of their business?
- What access security controls and data protection requirements are needed?>

Project Charter

1. Document Purpose

2. Problem Statement

3. Project Charter – Project

Project Name	
Author(s) of Charter	
Date of Submission	

4. Description

5. Key Project Personnel

6. Customers (Internal/External)

7. Scope

In Scope (Deliverables)	Out of Scope

8. Estimated Project Budget

Item	Description of Items	**Fiscal Funding** (one-time costs)	**Base Funding** (recurring costs)
	TOTALS:		

9. Assumptions

10. Constraints

11. Milestones

Milestone/Deliverable	Target Finish Date

12. Resources / Person Days Needed

Position	Project Responsibilities	Current or New	% Time	Person Days
TOTAL PDs:				

13. Risks

Risk Name	Description	Impact on Project (In Cost, PD, Schedule, Other)	Severity (H,M,L)	Probability (H,M,L)	Risk Rating

14. Success Criteria

Business Value	Criteria for Project Success	Measurement

15. Project Control

16. Consequences of failure

17. Alternatives

18. Contingency Plan

Business Case

1. **Project name**
2. **Origin / Background**
3. **Current state**
4. **Business objectives**
5. **Opportunities**
6. **Business strategic alignment**
7. **Value Proposition**
8. **Desired business outcomes**
9. **Outcomes roadmap**
10. **Business benefits (by outcome)**
11. **Quantified benefits value**
12. **Return on investment**
13. **Risks**
14. **Opportunity Cost**
15. **Solution scope**
16. **Assumptions**
17. **Constraints,**
18. **Options identified/ evaluated**

19. **Size, Scale and Complexity assessment**

20. **Deliverables**

21. **Organizational areas impacted (internally and externally),**

22. **Key stakeholders,**

23. **Dependencies**

24. **Approach**

25. **Workload estimate**

26. **Required resources – (Project leadership, Project governance, Resources)**

27. **Funding**

28. **Project controls**

29. **Reporting processes**

30. **Success Criteria**

Business Requirements

Requirement

Description -

Contact Information

Prepared by

Name, Title
Phone
Email address

Contributors

Name, Title, Company
Name, Title, Company

Primary Customer Contacts

Name Phone Number Email address	Name Phone Number Email Address

Distribution and Sign-Off

Area	Signature and Date	Area	Signature and Date

Overview

Give a brief outline explaining the current system and the system in use by the customer.

Summary of Impact

Explain what the impact will be on current processes or systems.

Proposed System

An overview of the customer requirements, include all phases of development.

Dependencies on Other Products

Describe any potential dependencies on other products or product lines.

Prerequisites or Stipulations for Receiving the Product

List any systems or products that will be required, hardware and software as well as versions.

Issues/Concerns/Constraints

List any concerns related to internal/external or 3rd party systems.

Future Considerations

Describe any possible future considerations, i.e. different configurations or systems supported.

GAP ANALYSIS

Item No.	End State	Current State	Gaps

REQUIREMENT CHANGE REQUEST

Requirement Change Request Form				
Change Request Date:	**Requestor's Name:**	**Requestor's Phone no:**	**Requestor's Fax no.:**	**Requestor's Email Address:**

Priority: (H/M/L)	**Requirement Name:**	**Attachments: (Y/N)**

Change request location:

Document Title:	**Version #:**	**Date:**

1. CHANGE

(Describe the change requested below and attach additional pages if necessary. Please describe what you were doing and what you want changed. Use a new form for each change requested.)

2. Analysis

Change Control Board (CCB):	Change Title:	Priority Assigned by CCB: (H, M, L)	Assigned To:

Error	Enhancement	Not in Error – Explain and Return to Originator	Insufficient Information for Analysis	Other
□	□	□	□	□

Explanation:

Approval for Proposed Change: (CCB USE ONLY)		
Approve	Disapprove	Defer
□	□	□

Authorized By:	Date:

Does this change have an external impact? (Yes/No)	Analysis Performed By:	Date:

3. CORRECTION

Describe change here and provide information:

Work Performed By:	**Date:**

4. RELEASE

(Provide documentation release information in this section)

Product:	**Release Date:**	**Version #:**

Work Performed By:	**Date:**

Impact Analysis Checklist for Requirements Changes

Implications of the Proposed Change

❑ Identify any existing requirements in the baseline that conflict with the proposed change.

❑ Identify any other pending requirement changes that conflict with the proposed change.

❑ What are the consequences of not making the change?

❑ What are possible adverse side effects or other risks of making the proposed change?

❑ Will the proposed change adversely affect performance requirements or other quality attributes?

❑ Will the change affect any system component that affects critical properties such as safety and security, or involve a product change that triggers recertification of any kind?

❑ Is the proposed change feasible within known technical constraints and current staff skills?

❑ Will the proposed change place unacceptable demands on any computer resources required for the development, test, or operating environments?

❑ Must any tools be acquired to implement and test the change?

❑ How will the proposed change affect the sequence, dependencies, effort, or duration of any tasks currently in the project plan?

❑ Will prototyping or other user input be required to verify the proposed change?

❑ How much effort that has already been invested in the project will be lost if this change is accepted?

❑ Will the proposed change cause an increase in product unit cost, such as by increasing third-party product licensing fees?

❑ Will the change affect any marketing, manufacturing, training, or customer support plans?

System Elements Affected by the Proposed Change

❑ Identify any user interface changes, additions, or deletions required.

❑ Identify any changes, additions, or deletions required in reports, databases, or data files.

❑ Identify the design components that must be created, modified, or deleted.

❑ Identify hardware components that must be added, altered, or deleted.

❑ Identify the source code files that must be created, modified, or deleted.

❑ Identify any changes required in build files.

❑ Identify existing unit, integration, system, and acceptance test cases that must be modified or deleted.

❑ Estimate the number of new unit, integration, system, and acceptance test cases that will be required.

❑ Identify any help screens, user manuals, training materials, or other documentation that must be created or modified.

❑ Identify any other systems, applications, libraries, or hardware components affected by the change.

❑ Identify any third party software that must be purchased.

❑ Identify any impact the proposed change will have on the project's software project management plan, software quality assurance plan, software configuration management plan, or other plans.

❑ Quantify any effects the proposed change will have on budgets of scarce resources, such as memory, processing power, network bandwidth, real-time schedule.

❑ Identify any impact the proposed change will have on fielded systems if the affected component is not perfectly backward compatible.

Effort Estimation for a Requirements Change

Effort **(Labor Hours)**	**Task**
________	Update the SRS or requirements database with the new requirement
________	Develop and evaluate prototype
________	Create new design components
________	Modify existing design components
________	Develop new user interface components
________	Modify existing user interface components
________	Develop new user publications and help screens
________	Modify existing user publications and help screens
________	Develop new source code
________	Modify existing source code
________	Purchase and integrate third party software
________	Identify, purchase, and integrate hardware components; qualify vendor
________	Modify build files
________	Develop new unit and integration tests
________	Modify existing unit and integration tests
________	Perform unit and integration testing after implementation
________	Write new system and acceptance test cases
________	Modify existing system and acceptance test cases
________	Modify automated test drivers
________	Perform regression testing at unit, integration, and system levels

________ Develop new reports

________ Modify existing reports

________ Develop new database elements

________ Modify existing database elements

________ Develop new data files

________ Modify existing data files

________ Modify various project plans

________ Update other documentation

________ Update requirements traceability matrix

________ Review modified work products

________ Perform rework following reviews and testing

________ Recertify product as being safe, secure, and compliant with standards.

________ Other additional tasks

________ **TOTAL ESTIMATED EFFORT**

Procedure:

1. Identify the subset of the above tasks that will have to be done.
2. Allocate resources to tasks.
3. Estimate effort required for pertinent tasks listed above, based on assigned resources.
4. Total the effort estimates.
5. Sequence tasks and identify predecessors.
6. Determine whether change is on the project's critical path.
7. Estimate schedule and cost impact.

Impact Analysis Report Template

Change Request ID: ______________

Title: ____

Description: ____

____

Analyst: __________________________

Date Prepared: __________________________

Prioritization Estimates:

Relative Benefit: ______ (1-9)

Relative Penalty: ______ (1-9)

Relative Cost: ______ (1-9)

Relative Risk: ______ (1-9)

Calculated Priority: ______(relative to other pending requirements)

Estimated total effort: ___________ labor hours

Estimated lost effort: ___________ labor hours (from discarded work)

Estimated schedule impact: ___________ days

Additional cost impact: ___________ dollars

Quality impact: ____

____

Other requirements affected: ____

____

Other tasks affected: ____

____

Integration issues: ____

Life cycle cost issues: ____

Other components to examine for possible changes:

____

____