Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA IT Specialization

IV Semester

MBSC-4.1G Software Project Management

Block 1

PREFACE

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products, . . . the list is almost endless. Software is virtually inescapable in a modern world. And as we move into the twenty-first century, it will become the driver for new advances in everything from elementary education to genetic engineering.

When a computer software succeeds—when it meets the needs of the people who use it, when it performs flawlessly over a long period of time, when it is easy to modify and even easier to use—it can and does change things for the better. But when software fails—when its users are dissatisfied, when it is error prone, when it is difficult to change and even harder to use—bad things can and do happen. We all want to build software that makes things better, avoiding the bad things that lurk in the shadow of failed efforts. To succeed, we need discipline when software is designed and built. We need an engineering approach.

The whole material is organized into four modules each with four units. Each unit lists the objectives of the study along with the relevant questions, illustrations and suggested reading to better understand the concepts.

Wish you happy reading!!!

KARNATAKA STATE OPEN UNIVERSITY

MUKTHAGANGOTRI, MYSURU-06

Dept. of Studies and Research in Management

MBA IT

IV SEMESTER

MBSC-4.1G Software Project Management

BLOCK 1: Introduction to Software Project Management

BLOCK 1 INTRODUCTION

A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal. Projects can vary from simple to difficult and can be operated by one person or a hundred.

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline in which software projects are planned, implemented, monitored and controlled.

Software is a non-physical product. Software development is a new stream in business and there is very little experience in building software products. Most of the software products are made to fit clients' requirements. The most important is that the basic technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. Such type of business and environmental constraints increase risk in software development hence it is essential to manage software projects efficiently. It is necessary for an organization to deliver quality products, keep the cost within the client's budget constrain and deliver the project as per schedule. Hence in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

This block consists of four units and is organized as follows:

Unit 1 speaks about the past and the present Software Project Management methodology and their pros and cons, the different phases and the process models of Software Projects.

Unit 2 focuses on the context of Information Technology in Software Projects.

Unit 3 emphasises on the major parameters of Software Project Management like People, Product, Process, the Software Configuration Management and Testing Strategies.

Unit 4 highlights the Quality Management concepts in Software Project Management. The different concepts, review techniques and quality assurance are underlined in this unit.

UNIT-1: INTRODUCTION TO PROJECT MANAGEMENT

STRUCTURE

1.0 OBJECTIVES

After studying this unit, you will be able to:

- Understand the process and the need of SPM
- Explain the difference between the past and present system
- Recognize the different phases of project life cycle
- Discuss various Software Process Models

1.1 INTRODUCTION

In this unit, we are going to discuss about project management. Initially, past and the present Software Project Management methodology and their pros and cons are discussed. Further, conventional management types, the different phases and the process models of Software Projects are discussed in detail.

1.2 INTRODUCTION TO PROJECT MODEL

Project management is the discipline of defining and achieving project goals while optimizing for any resource constraints during the lifecycle of a project.

A subset of project management, Software Project Management is the practice of planning and delivering software development projects within variables such as:

- Time
- Quality
- Cost
- The wider scope

Before getting into the process of Project Management, one has to be clearly aware of Software Projects. A *project* is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

Regardless, every project must have the following components:

- Goal: What are you trying to achieve?
- Timeline: When are you trying to achieve it by?

- Budget: How much will it cost to achieve?
- Stakeholders: Who are the major players who have an interest in this project?
- Project manager: Who is going to make sure everything that needs to be completed gets completed?

A project is not something routine. Day-to-day operations or maintenance is not considered a project because it does not have a definitive start and end.

Project management is the practice of applying knowledge, skills, tools, and techniques to complete a project according to specific requirements. Understanding project management comes down to identifying the problem, creating a plan to solve the problem, and then executing on that plan until the problem has been solved. That may sound simple, but there is a lot that goes into it at every stage of the process.

1.3 PROJECT MANAGEMENT: PAST AND PRESENT

Project management, as an idea, goes back a very long way. Thinking about all of the things that have been built in the history of civilization, there are thousands of years of project experience to learn from. A dotted line can be drawn from the software developers of today back through time to the builders of the Egyptian pyramids or the architects of the Roman aqueducts. For their respective eras, project managers have played similar roles, applying technology to the relevant problems of the times. Yet today, when most people try to improve how their web and software development projects are managed, it's rare that they pay attention to lessons learned from the past. The timeline we use as the scope for useful knowledge is much closer to present day than it should be.

Four Periods in the Development of Modern Project Management

[1] Prior to 1958: Craft system to human relations. During this time, the evolution of technology, such as, automobiles and telecommunications shortened the project schedule. For instance, automobiles allowed effective resource allocation and mobility, whilst the telecommunication system increased the speed of communication. Furthermore, the job specification which later became the basis of developing the Work Breakdown Structure (WBS) was widely used and Henry Gantt invented the Gantt chart. Examples of projects undertaken during this period as supported by documented evidence include: (a) Building the Pacific Railroad in 1850s; (b) Construction of the Hoover Dam in 1931-1936, that employed approximately 5,200 workers and is still one of the highest gravity dams in the U.S. generating about four billion kilowatt hours a year; and (c) The Manhattan Project in 1942-

1945 that was the pioneer research and development project for producing the atomic bomb, involving 125,000 workers and costing nearly $2 billion.

[2] 1958-1979: Application of Management Science. Significant technology advancement took place between 1958 and 1979, such as, the first automatic plain-paper copier by Xerox in 1959. Between 1956 and 1958 several core project management tools including CPM and PERT were introduced. However, this period was characterised by the rapid development of computer technology. The progression from the mainframe to the mini-computer in the 1970s made computers affordable to medium size companies. In 1975, Bill Gates and Paul Allen founded Microsoft. Furthermore, the evolution of computer technology facilitated the emergence of several project management software companies, including, Artemis (1977), Oracle (1977), and Scitor Corporation (1979). In the 1970s other project management tools such as Material Requirements Planning (MRP) were also introduced.

[3] 1980-1994: Production Centre Human Resources. The 1980s and 1990s are characterised by the revolutionary development in the information management sector with the introduction of the personal computer (PC) and associated computer communications networking facilities. This development resulted in having low cost multitasking PCs that had high efficiency in managing and controlling complex project schedules. During this period low cost project management software for PCs became widely available that made project management techniques more easily accessible.

[4] 1995-Present: Creating a New Environment. This period is dominated by the developments related to the Internet that changed dramatically business practices in the mid 1990s. The Internet has provided fast, interactive, and customised new medium that allows people to browse, purchase, and track products and services online instantly. This has resulted in making firms more productive, more efficient, and more client oriented. Furthermore, many of today's project management software have an Internet connectivity feature. This allows automatic uploading of data so that anyone around the globe with a standard browser can: (a) input the most recent status of their assigned tasks; (b) find out how the overall project is doing; (c) be informed of any delays or advances in the schedule; and (d) stay "in the loop" for their project role, while working independently at a remote site.

Current and Emergent Practices in Project Management

Between 2010 and 2020, we expanded our understanding of project management. PM's field is diffuse and multi-disciplinary and offers a considerable body of literature in related areas

like agile project management, strategy execution, business analysis, and more. There are various methods and frameworks, and we realized that we could build our hybrid methods. But how?

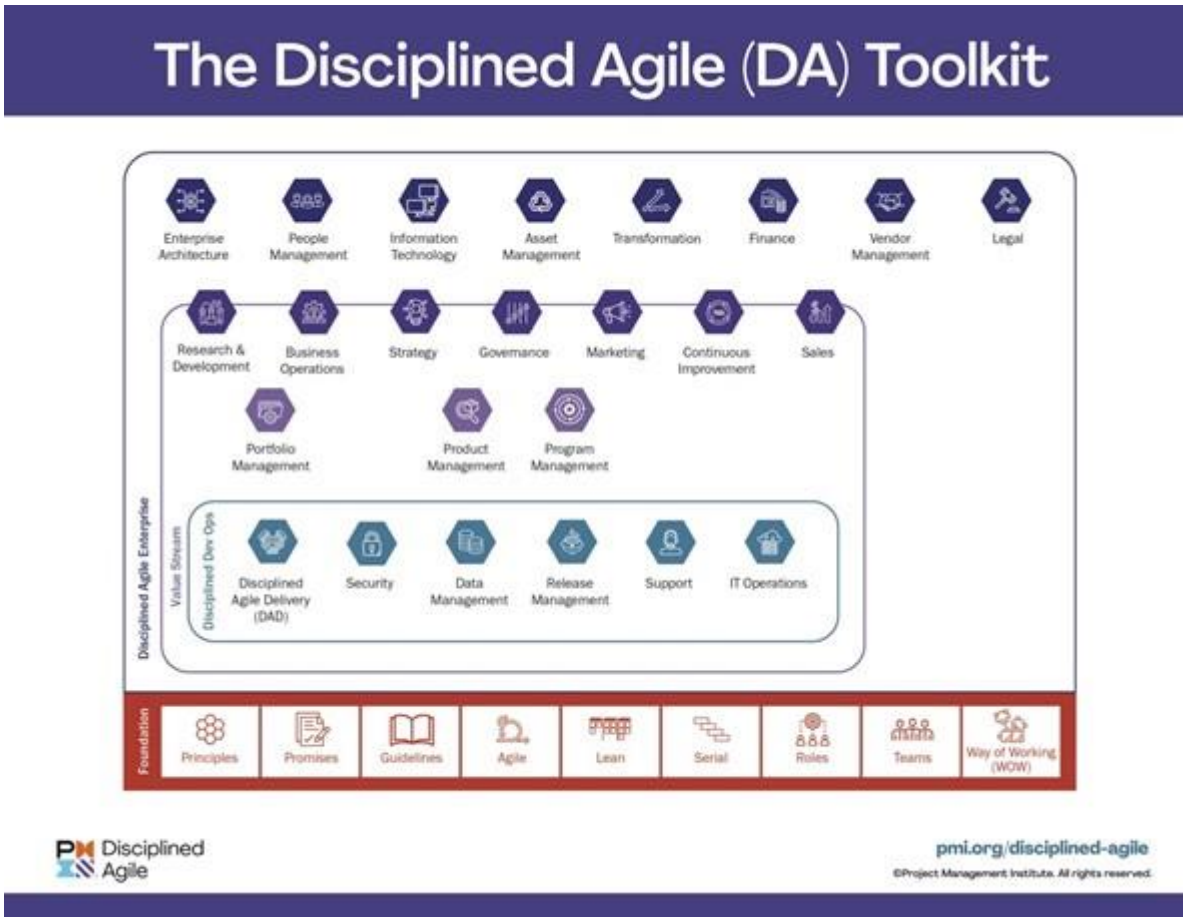


Figure 1.1 – Disciplined agile toolkit (PMI)

Disciplined Agile (Figure 1.2) is a toolkit to help people and organizations to fully and truly become agile by combining different execution approaches, including serial and agile, to the other business processes in an integrated fashion. Describing DA in detail is beyond this article's scope, but that leads us to "What's next?"

1.4 PROJECT MANAGEMENT OVERVIEW

Project management is the use of specific knowledge, skills, tools and techniques to deliver something of value to people. The development of software for an improved business process, the construction of a building, the relief effort after a natural disaster, the expansion of sales into a new geographic market—these are all examples of projects.

All projects are a temporary effort to create value through a unique product, service or result. All projects have a beginning and an end. They have a team, a budget, a schedule and a set of expectations the team needs to meet. Each project is unique and differs from routine operations—the ongoing activities of an organization—because projects reach a conclusion once the goal is achieved.

The changing nature of work due to technological advances, globalization and other factors means that, increasingly, work is organized around projects with teams being brought together based on the skills needed for specific tasks.

Leading these projects are Project Professionals—people who either intentionally or by circumstance are asked to ensure that a project team meets its goals. Project professionals use many different tools, techniques and approaches to meet the needs of a project.

Some projects are needed to quickly resolve problems, with an understanding that improvements will be made over a period of time. Other projects have a longer duration and/or produce a product or other outcome that will not need major improvements outside of projected maintenance, such as a highway.

Still others will be a mix of both of these types of projects. Project professionals use a variety of skills and knowledge to engage and motivate others to reach a project's goals. Project professionals are critical to the success of projects and are highly sought after to help organizations achieve their goals.

1.4.1 PROJECT CHARACTERISTICS

When considering whether or not you have a project on your hands, there are some things to keep in mind. First, is it a project or an ongoing operation? Second, if it is a project, who are the stakeholders? And third, what characteristics distinguish this endeavor as a project?

Projects have several characteristics:

- Projects are unique.
- Projects are temporary in nature.
- Projects have a definite beginning and ending date.
- Projects are completed when the project goals are achieved or it's determined the project is no longer viable.

A successful project is one that meets or exceeds the expectations of the stakeholders.

Project Management is not about managing people alone. Project management can be divided into different process groups and knowledge areas. Process groups include initiating, planning, executing, monitoring and controlling, and closing. Knowledge areas include integration, scope, time cost, quality, human resources, communication, risk, procurement and stakeholder management.

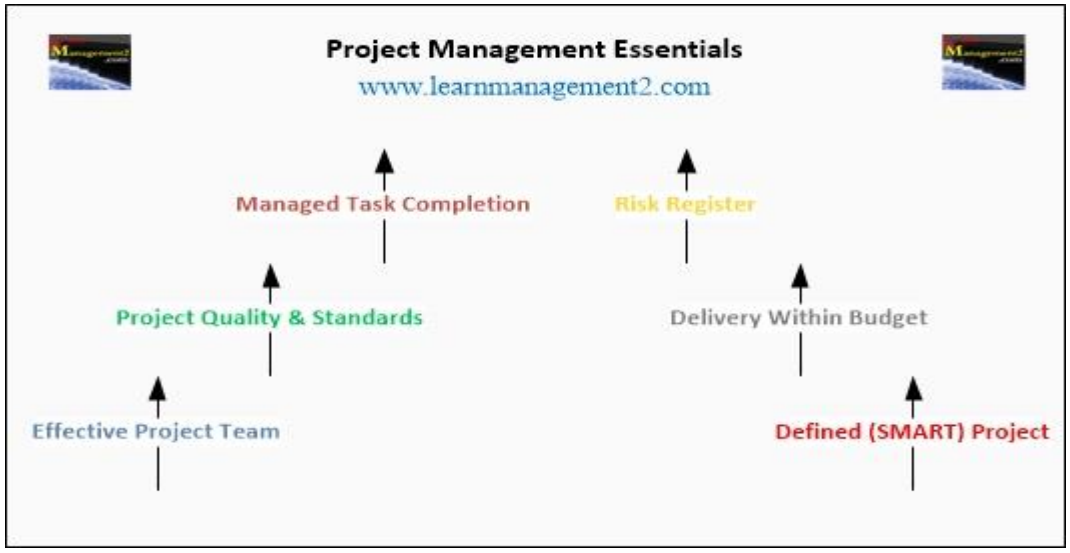


Figure 1.2 – Project Management Essentials

1.4.2 PROJECT DEADLINES AND PENALTIES

Often projects need to be completed by a set date and there is no leeway for example the tasks for the birthday party will need to be completed by the day of the birthday. Other projects may incur penalties if they are not completed on time for example the costs of flights will increase as you get closer to the date of the flight.

1.4.3 PROJECT BUDGETS

The majority of projects have a budget for completing the project tasks. Using the decorating a house example, there are many ways to decorate a house; some options will be more expensive than others. Your final choice of décor will depend on how much money you have to buy paint, wallpaper, tools etc.

1.4.4 PROJECT RISK REGISTER

A risk register contains a list of things which may hinder the project. The purpose of the risk register is to identify which risks need to be actively managed in order to

1. Prevent the risk occurring and/or
2. To minimise the impact of a risk if its occurrence can't be prevented.

As the list of project risks could be quite long the project team may decide that it is impractical to try and manage all of the risks; instead they may focus their attention on the risks that are most likely to occur and the ones that will have the greatest impact on the project.

1.4.5 PROJECT MANAGEMENT DEFINITION

In each of our examples there are:

1. A number of tasks to complete
2. A date by which the tasks need to be completed
3. A budget within which the project needs to be completed

Bearing each of these requirements in mind let's define project management. Project management is about ensuring that the tasks of a project are completed on time and within budget.

1.4.6 PROJECT QUALITY AND STANDARDS

The standard to which the tasks are completed is very important, otherwise the project may cause more problems than benefits. For example a badly decorated house will need to be redecorated and a holiday that people do not enjoy will cause people stress and not relaxation.

There are a number of ways to manage a project but each method should state who is responsible for each task and set a deadline for the completion of the task. The project manager should regularly review how each task is progressing and take action if it is not on track for completion by the deadline.

1.5 SOFTWARE PROJECT VS OTHER TYPES

Invisibility: When a physical artifact such as a bridge is constructed the progress can actually be seen. With software, progress is not immediately visible. Software project management can be the process of making the invisible visible.

Complexity: Per dollar, pound or euro spent, software products contain more complexity than other engineered artifacts.

Conformity: The 'traditional' engineer usually works with physical systems and materials like cement and steel. These physical systems have complexity, but are governed by consistent physical laws. Software developers have to conform to the requirement of human clients. It is

not just that individuals can be inconsistent. Organizations, because of lapses in collective memory, in internal communication or in effective decision making, can exhibit remarkable, ‘organizational-stupidity'.

Flexibility: That software is easy to change is seen as a strength. However, where the software system interfaces with a physical or organizational system, it is accommodate the other components rather than vice versa. Thus software systems are particularly subject to change.

1.6 SOFTWARE PROJECT LIFECYCLE (PHASES)

With all the complex processes involved in software development, it’s easy to forget the fundamental process for a successful **software development life cycle (SDLC)**. The SDLC process includes planning, designing, developing, testing and deploying with ongoing maintenance to create and manage applications efficiently. When faced with the task of producing high-quality software that meets a client’s expectations, requirements, time-frame, and cost estimations; understanding the SDLC is crucial.

“SDLC methodologies” are used to create complex applications of varying sizes and scales, such as Agile, Waterfall and Spiral. Each model follows a particular life cycle in order to ensure success in the process of software development

1.6.1 Requirement Collection: - The first phase of the software development life cycle process is requirement collection, where a business analyst will collect the business needs of the customer in the form of requirement documents and planning for the quality assurance requirements.

This stage gives a clear picture of the scope of the entire project and the anticipated issues, opportunities, and directives, which triggered the project and need teams to get detailed and precise requirements. It will help companies to finalize the necessary timeline to finish the work of that system.

1.6.2 Feasibility Study: - In the second stage of the software development life cycle, based on the requirements, a set of people sit and analysis if the project is Doable or not Doable, which means that organization has enough resource, cost, time so that they can deliver the product on time with the high -quality product. We can check the feasibility in a different manner, i.e., Economic, Legal, Operation, feasibility, Technical, Schedule.

1.6.3 Design: - In the third stage of the software development life cycle, once the feasibility is done, prepare a blueprint of the application. The blueprint has flow charts, flow diagrams, and decision tree and user interface components.

The Designer will design the document in two ways:

HLD (High-Level Design): In HLD, we have a brief description, name of each module, interface relationship, dependencies between modules, database tables, and complete architecture diagrams

LLD (Low-Level Design): In LLD, we have functional logic of the modules, Database tables, which include type and size, pointing all kinds of dependency issues and listing of error messages

1.6.4 Coding: - The fourth stage of the software development life cycle is coding, after the complication of requirements and designing phase, the developer starts writing code using the particular program language and, the developer needs to follow specific predefined coding guidelines and used programming tools like compiler, interpreters, debugger to generate and implement the code. Coding is the time taking process of the Software Development Life Cycle process

1.6.5 Testing: - In the testing phase ofthe software development life cycle, once the coding is completed, the application is handed over to the test engineers where they start checking the functionality of an application according to the requirement. During the testing process, we may encounter some bugs which need to be fixed by developers and retested by the test engineers.

1.6.6 Installation: - In the installation phase, the process will continue until the application is bug-free/ stable/ and works according to customer needs. The stable application is handed over to the customer in the form of installation or deploying or rolls out.

1.6.7 Maintenance: - The last phase of software development life cycle, when a customer starts using the software they may place some issues which need to be in-detail tested & fixed and handover back to the customer and bug fixes, up-gradation, enhancement is done under maintenance phase.

“All this phase is essential to make a good quality product or an application. Without completion of any phase, it might not deliver a good quality product because all the phases are connected.”

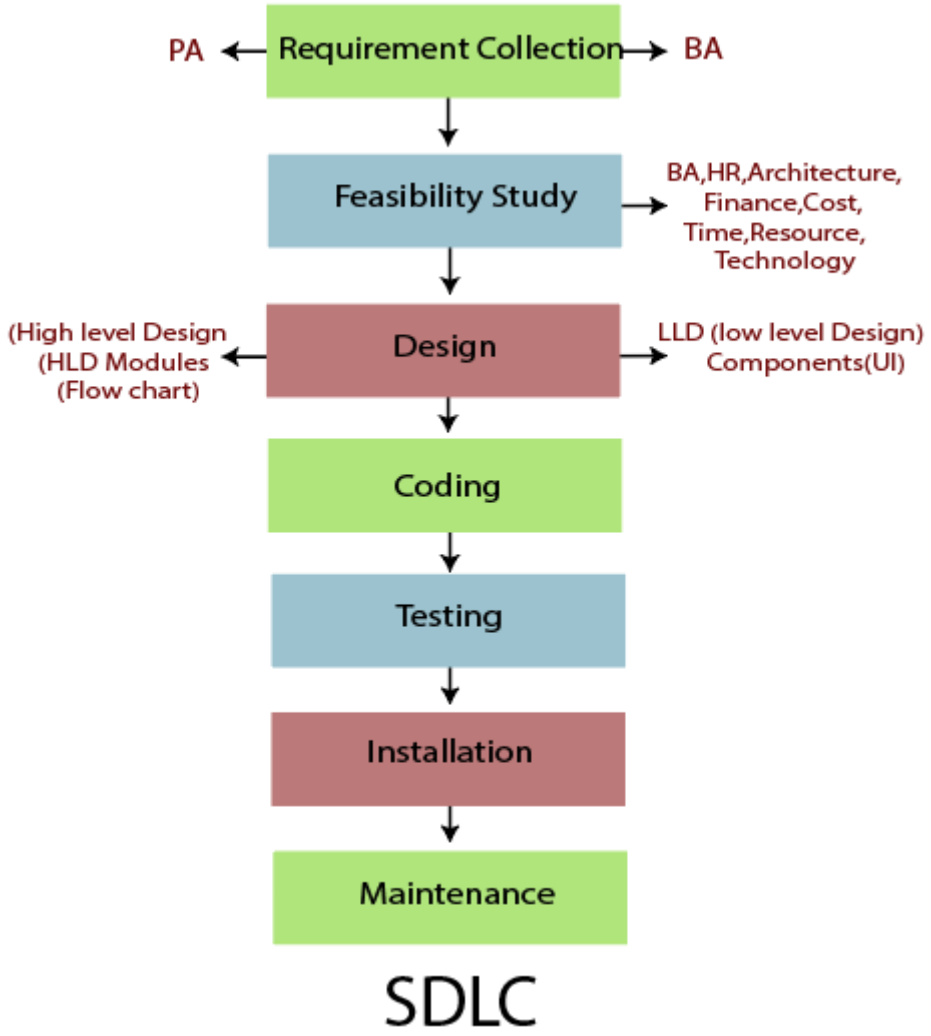


Figure 1.3: Phases of Software Development Life Cycle

1.7 SOFTWARE PROCESS MODELS

Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:

- Specification – defining what the system should do;
- Design and implementation – defining the organization of the system and implementing the system;
- Validation – checking that it does what the customer wants;
- Evolution – changing the system in response to changing customer needs.

Types of Software Process Model

Software processes, methodologies and frameworks range from specific prescriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a “sponsor” or “maintenance” organization distributes an official set of documents that describe the process.

Software Process and Software Development Lifecycle Model

One of the basic notions of the software development process is SDLC models which stands for Software Development Life Cycle models. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:

- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model
- Iterative model
- Spiral model
- Prototype model

1.7.1 WATERFALL MODEL

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks. The approach is typical for certain areas of engineering design.

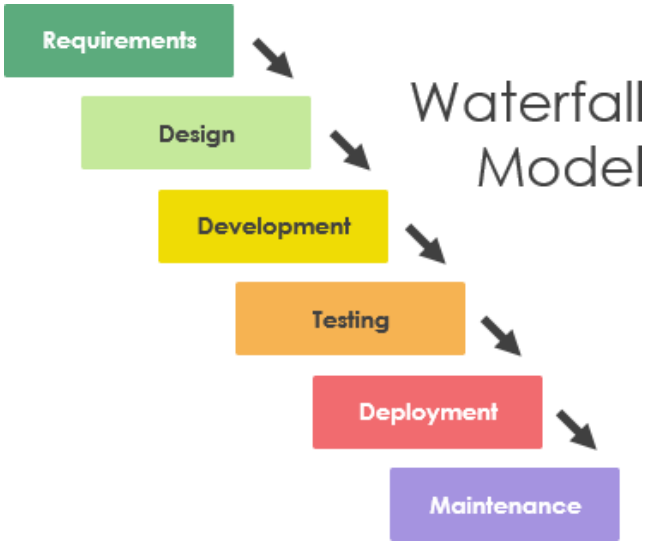


Figure 1.4: Waterfall Model

1.7.2 V MODEL

The V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

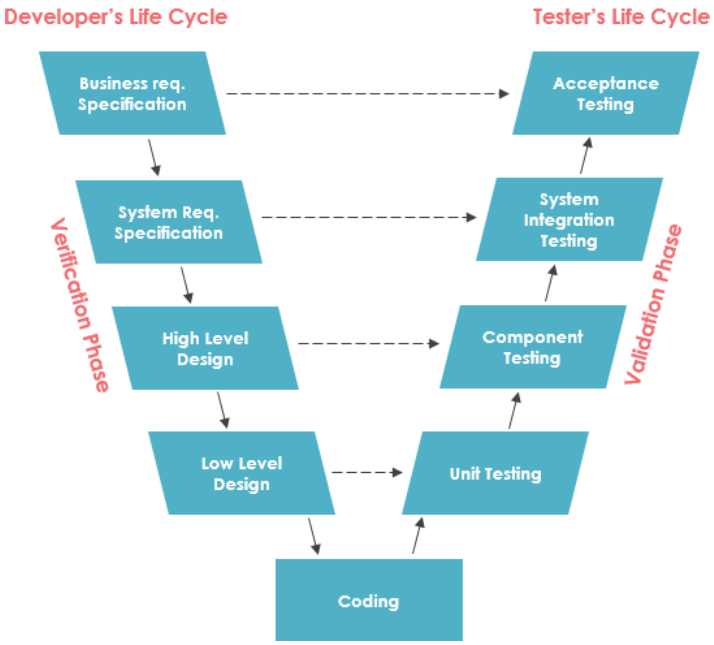


Figure 1.5: V Model

1.7.3 INCREMENTAL MODEL

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionally has been implemented. This model combines the elements of the waterfall model with the iterative philosophy of prototyping.

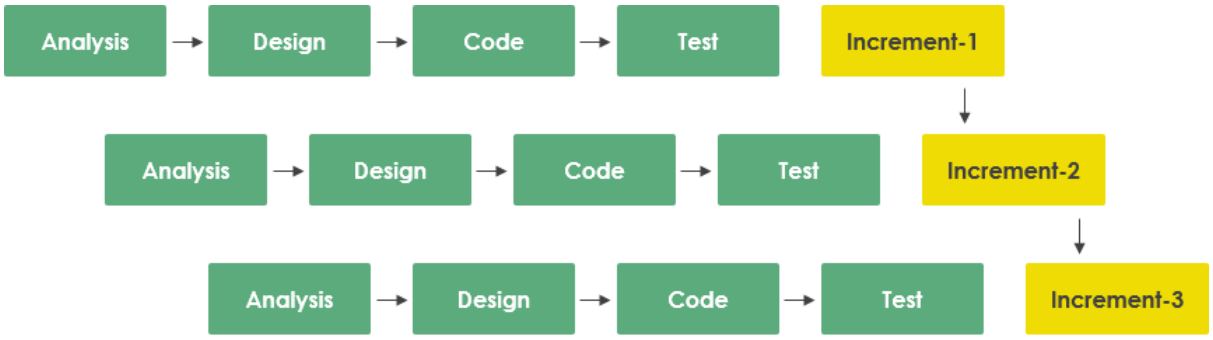


Figure 1.6: Incremental Model

1.7.4 ITERATIVE MODEL

An iterative life cycle model does not attempt to start with a full specification of requirements by first focusing on an initial, simplified set user features, which then progressively gains more complexity and a broader set of features until the targeted system is complete. When adopting the iterative approach, the philosophy of incremental development will also often be used liberally and interchangeably.

In other words, the iterative approach begins by specifying and implementing just part of the software, which can then be reviewed and prioritized in order to identify further requirements. This iterative process is then repeated by delivering a new version of the software for each iteration. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal software specification may also be required.

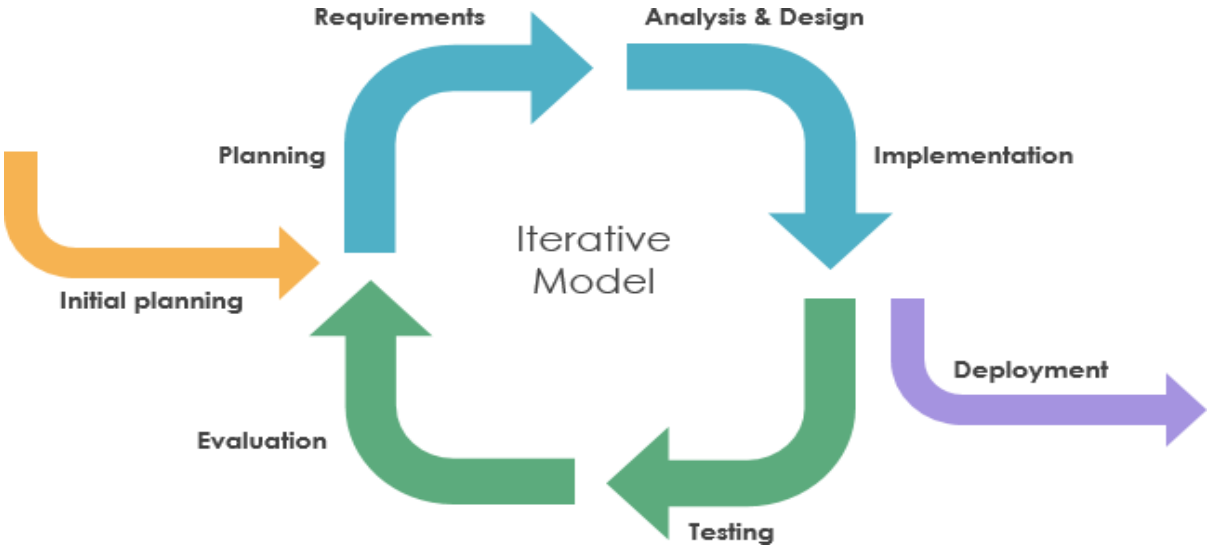


Figure 1.7: Iterative Model

1.7.5 RAD MODEL

Rapid application development was a response to plan-driven waterfall processes, developed in the 1970s and 1980s, such as the Structured Systems Analysis and Design Method (SSADM). Rapid application development (RAD) is often referred as the adaptive software development. RAD is an incremental prototyping approach to software development that end users can produce better feedback when examining a live system, as opposed to working strictly with documentation. It puts less emphasis on planning and more emphasis on an adaptive process.

RAD may resulted in a lower level of rejection when the application is placed into production, but this success most often comes at the expense of a dramatic overruns in project costs and schedule. RAD approach is especially well suited for developing software that is driven by user interface requirements. Thus, some GUI builders are often called rapid application development tools.

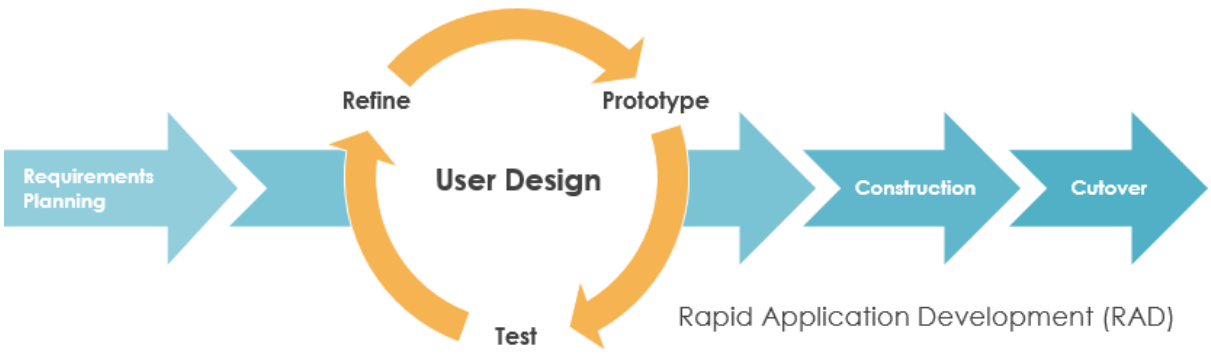


Figure 1.8: RAD Model

1.7.6 SPIRAL MODEL

The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model. A spiral model looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in loops. Each loop of the spiral is called a Phase of the software development process.

The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.

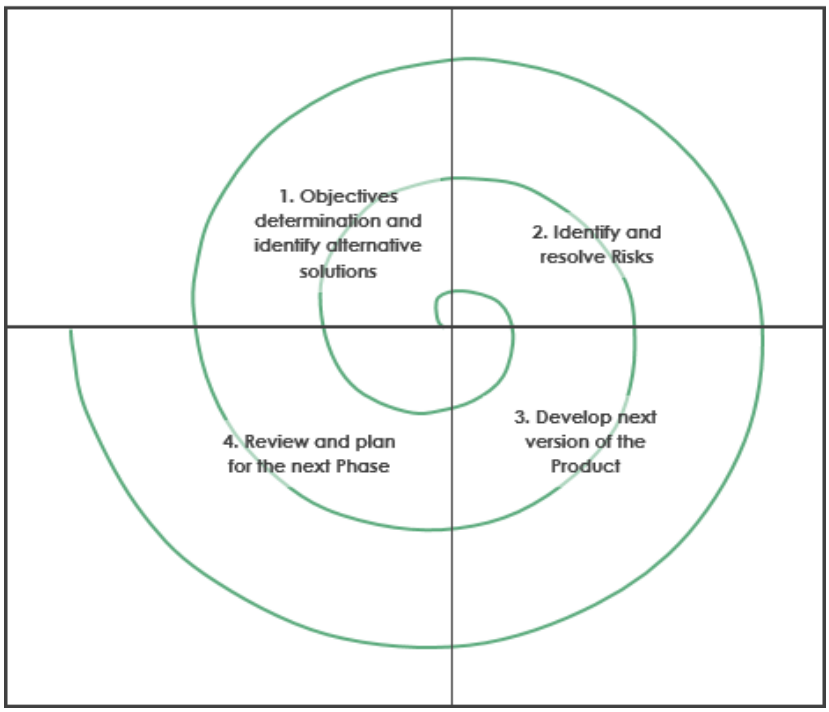


Figure 1.9: Spiral Model

1.7.7 AGILE MODEL

Agile is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto that is a way of thinking that enables teams and businesses to innovate, quickly respond to changing demand, while mitigating risk. Organizations can be agile using many of the available frameworks available such as Scrum, Kanban, Lean, Extreme Programming (XP) and etc.

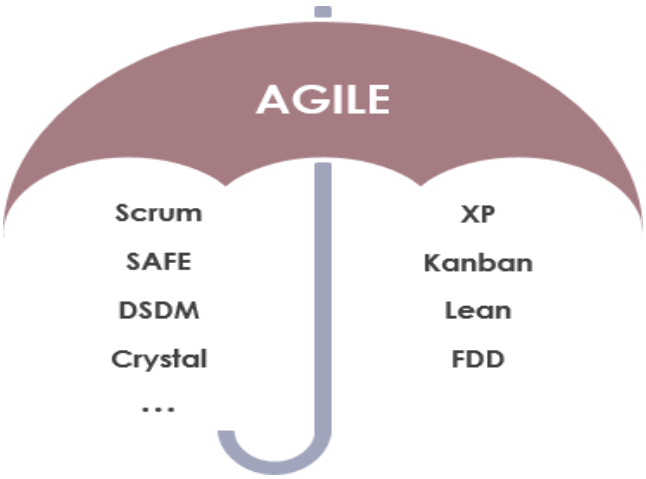


Figure 1.10: Agile Model

The Agile movement proposes alternatives to traditional project management. Agile approaches are typically used in software development to help businesses respond to unpredictability which refer to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

The primary goal of being Agile is empowered the development team the ability to create and respond to change in order to succeed in an uncertain and turbulent environment. Agile software development approach is typically operated in rapid and small cycles. This results in more frequent incremental releases with each release building on previous functionality. Thorough testing is done to ensure that software quality is maintained.

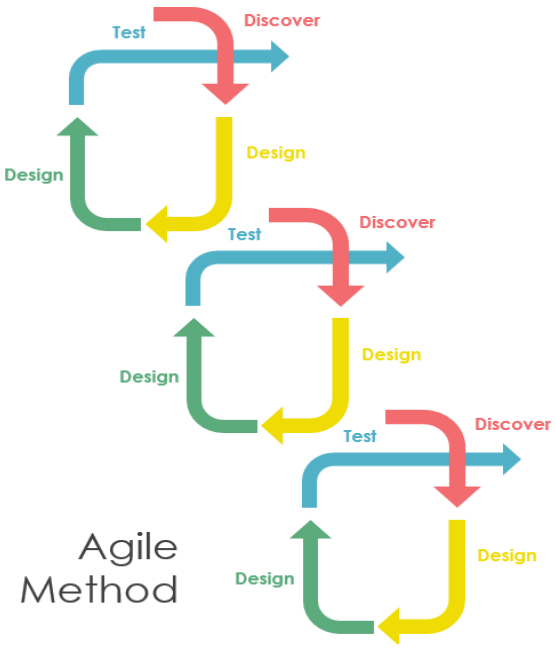


Figure 1.11: Agile Method

1.8 CHECK YOUR PROGRESS

1. What is the objective of Software Project Management?
2. What is Software Process Model?
3. PM's field is diffuse and multi-disciplinary and offers a considerable body of literature in related areas like ----------, ----------, ----------, and more.
4. What is Conventional Software Management?
5. The exact number of phases needed to develop the product can be ---------- the project

Answers to check your progress:

1. Software project management focuses on developing a product that will have a positive effect on an organization. Without project management, a software development team may begin working on a project without any clear vision or guidance, resulting in more frequent errors and confusion.
2. In software engineering, a software process model is the mechanism of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle.
3. like agile project management, strategy execution, business analysis
4. In the past, organizations used conventional software management. This management utilized custom tools and process and virtually custom components built-in primitive languages. Thus, the performance of the project was very much predictable in the schedule, cost, and quality. It is a practically outdated technique and technology.
5. Varied

1.9 SUMMARY

In this Unit, past and the present Software Project Management methodology and their pros and cons are discussed, conventional management types, the different phases and the process models of Software Projects. **Unit 2** focuses on the context of Information Technology in Software Projects. **Unit 3** emphasises on the major parameters of Software Project Management like People, Product, Process, the Software Configuration Management and Testing Strategies. **Unit 4** highlights the Quality Management concepts in Software Project Management. The different concepts, review techniques and quality assurance are underlined in this unit.

1.10 KEYWORDS

- **Project:** It is well-defined task, which is a collection of several operations done in order to achieve a goal.
- **Project management:** It is the practice of applying knowledge, skills, tools, and techniques to complete a project according to specific requirements.
- **Coding:** It is a computer programming language that helps to communicate with a computer.
- **Software maintenance**: It is the process of changing, modifying, and updating software to keep up with customer needs.
- **Software Processes:** It is a coherent set of activities for specifying, designing, implementing and testing software systems.

1.11 QUESTIONS FOR SELF STUDY

1. The difference between the Project Management followed in the past and present.
2. Write an overview on the project management overview and its characteristics.
3. Describe the phases of Software Development Life Cycle.
4. Write the principles of Conventional and Current Software Management.
5. The difference and purpose of various software process models along with their diagrams.
6. What kind of evolution practices are followed in Software Economics?

1.12 REFERENCES

1. https://www.projectmanagement.com/
2. https://www.oreilly.com/
3. https://www.projectsmart.co.uk/
4. https://pressbooks.bccampus.ca/
5. https://www.pmi.org/
6. https://www.learnmanagement2.com/
7. https://www.tutorialandexample.com/
8. https://www.geeksforgeeks.org/
9. https://www.visual-paradigm.com/guide/software-development-process/what-is-a-software-process-model/

UNIT-2: TECHNOLOGY CONTEXT

STRUCTURE

2.0 OBJECTIVES

After studying this unit, you will be able to:

- Understand the systems view of project management and how it applies to information technology projects
- Analyze a formal organization using the structural, human resources, political, and symbolic organizational frames
- Describe the differences among functional, matrix, and project organizational structures
- Explain why stakeholder management and top management commitment are critical for a project's success
- Understand the concept, development, implementation, and close-out phases of the project life cycle
- Distinguish between project development and product development
- Discuss the unique attributes and diverse nature of information technology projects
- List the skills and attributes of a good project manager in general and in the information technology field

2.1 INTRODUCTION

In this unit, we are going to discuss about systems view of project management and the application of Information Technology. The analysis helps the organization to provide the perspectives of structural, hr and political frames. This also helps to understand the conceptual, developmental, implementation and close-out phases of project life cycle. It also provides the clarity between project and product development.

2.2 A SYSTEM VIEW OF PROJECT MANAGEMENT

Even though projects are temporary and intended to provide a unique product or service, you cannot run projects in isolation. If project managers lead projects in isolation, it is unlikely that they will ever truly serve the needs of the organization. Therefore, projects must operate in a broad organizational environment, and project managers need to consider projects within the greater organizational context. To handle complex situations effectively, project managers need to take a holistic view of a project and understand how it relates to the larger organization. Systems thinking describe this holistic view of carrying out projects within the context of the organization.

The term **systems approach** emerged in the 1950s to describe a holistic and analytical approach to solving complex problems that includes using a systems philosophy, systems analysis, and systems management. Systems are sets of interacting components that work within an environment to fulfil some purpose. For example, the human body is a system composed of many subsystems, including the nervous system, the skeletal system, the circulatory system, and the digestive system. Organizations are also systems, with people in various roles working together to design, develop, deliver, and sell various products and services. A **systems philosophy** is an overall model for thinking about things as systems.

Systems analysis is a problem-solving approach that requires defining the scope of the system, dividing it into components, and then identifying and evaluating its problems, opportunities, constraints, and needs. Once this is completed, the systems analyst then examines alternative solutions for improving the current situation; identifies an optimum, or at least satisfactory, solution or action plan; and examines that plan against the entire system. Systems management addresses the business, technological, and organizational issues associated with creating, maintaining, and modifying a system.

Using a systems approach is critical to successful project management. If top management and project managers are to understand how projects relate to the whole organization, they must follow a systems philosophy. They must use systems analysis to address needs with a problem-solving approach. They must use systems management to identify key issues in business, technological, and organizational spheres related to each project in order to identify and satisfy key stakeholders and do what is best for the entire organization.

2.3 UNDERSTANDING ORGANIZATIONS

The systems approach requires that project managers always view their projects in the context of the larger organization. Organizational issues are often the most difficult part of working on and managing projects. In fact, many people believe that most projects fail because of organizational issues like company politics. Project managers often do not spend enough time identifying all the stakeholders involved in projects, especially the people opposed to the projects. Also, project managers often do not spend enough time considering the political context of a project or the culture of the organization. To improve the success rate of IT projects, it is important for project managers to develop a better understanding of people as well as organizations.

THE FOUR FRAMES OF ORGANIZATIONS

As shown in Figure 2.1, you can try to understand organizations better by focusing on different perspectives. Organizations can be viewed as having four different frames: structural, human resources, political, and symbolic.1

- The **structural frame** deals with how the organization is structured (usually depicted in an organizational chart) and focuses on different groups' roles and responsibilities to meet the goals and policies set by top management. This frame is very rational and focuses on coordination and control. For example, within the structural frame, a key IT issue is whether a company should centralize the IT personnel in one department or decentralize across several departments. You will learn more about organizational structures in the next section.
- The **human resources (HR) frame focuses** on producing harmony between the needs of the organization and the needs of people. It recognizes that mismatches can occur between the needs of the organization and those of individuals and groups, and works to resolve any potential problems. For example, many projects might be more efficient for

the organization if employees worked 80 or more hours a week for several months. However, this work schedule would conflict with the personal lives and health of many employees. Important IT issues related to the human resources frame are the shortage of skilled IT workers within the organization and unrealistic schedules imposed on many projects.

Structural frame: Roles and responsibilities, coordination, and control. Organizational charts help describe this frame.	Human resources frame: Providing harmony between needs of the organization and needs of people.
Political frame: Coalitions composed of varied individuals and interest groups. Conflict and power are key issues.	Symbolic frame: Symbols and meanings related to events. Culture, language, traditions, and image are all parts of this frame.

Figure 2.1 – Perspective on Organizations

- The **political frame** addresses organizational and personal politics. Politics in organizations take the form of competition among groups or individuals for power, resources, and leadership. The political frame emphasizes that organizations are coalitions composed of varied individuals and interest groups. Often, important decisions need to be made about the allocation of scarce resources. Competition for resources makes conflict a central issue in organizations, and power improves the ability to obtain those resources. Project managers must pay attention to politics and power if they are to be effective. It is important to know who opposes your projects as well as who supports them. Important IT issues related to the political frame are the differences in power between central functions and operating units or between functional managers and project managers.
- The **symbolic frame** focuses on symbols and meanings. In this frame, the most important aspect of any event in an organization is not what actually happened, but what it means. Was it a good sign that the CEO came to a kick-off meeting for a project, or was it a threat? The symbolic frame also relates to the company's culture. How do people dress? How many hours do they work? How do they run meetings? Many IT projects are international and include stakeholders from various cultures. Understanding those cultures is also a crucial part of the symbolic frame.

ORGANIZATIONAL STRUCTURES

Many discussions of organizations focus on their structure. Three general classifications of organizational structures are functional, project, and matrix. Many companies today use all

three structures somewhere in the organization, but using one is most common. Figure 2.2 portrays the three organizational structures. A **functional organizational structure** is the hierarchy most people think of when picturing an organizational chart. Functional managers or vice presidents in specialties such as engineering, manufacturing, IT, and human resources report to the chief executive officer (CEO). Their staffs have specialized skills in their respective disciplines. For example, most colleges and universities have very strong functional organizations. Only faculty members in the business department teach business courses; faculty in the history department teach history; faculty in the art department teach art, and so on.

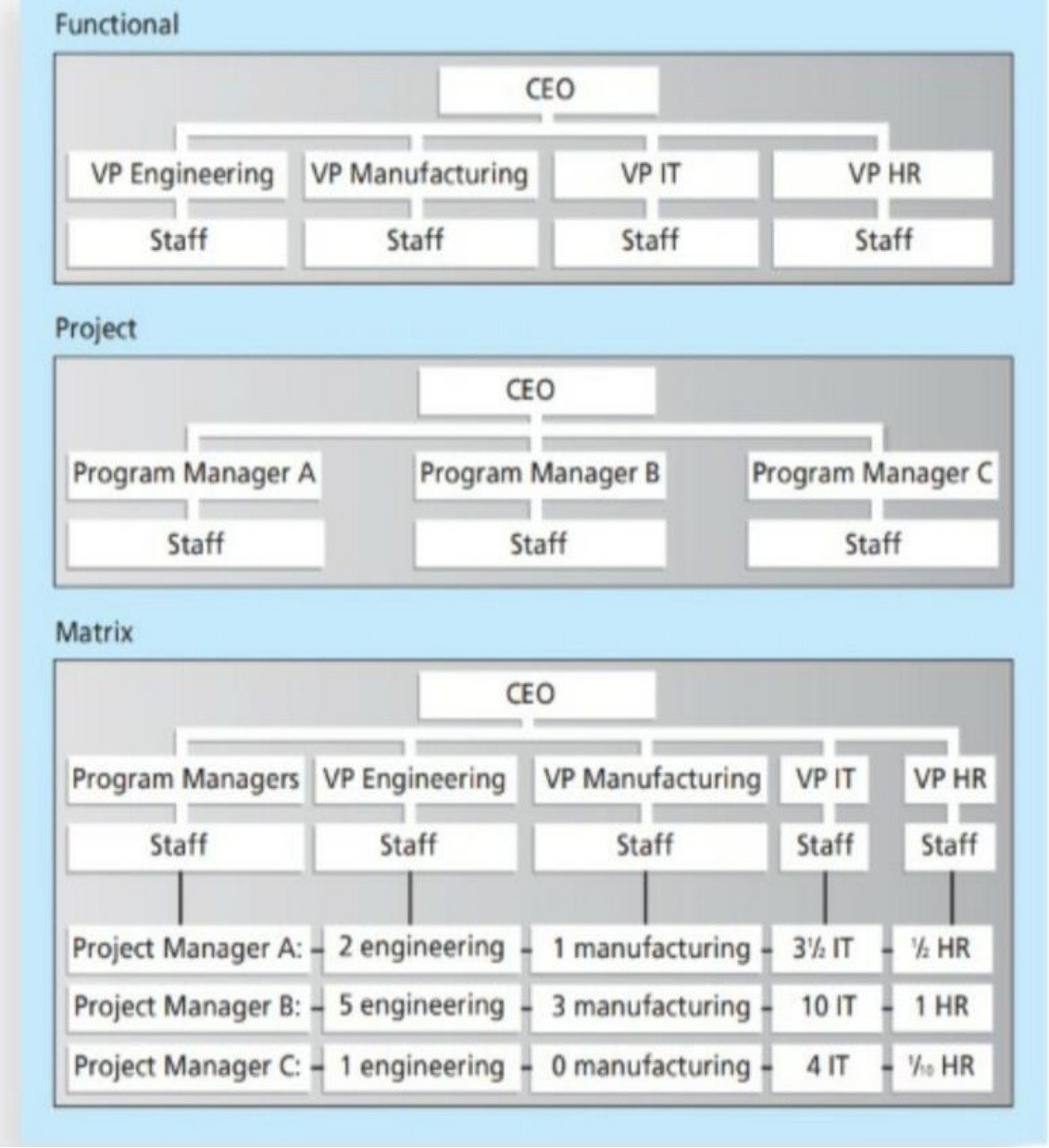


Figure 2.2 – Funtional, project and matrix organizational structures

A **project organizational structure** also is hierarchical, but instead of functional managers or vice presidents reporting to the CEO, program managers report to the CEO. Their staffs have a variety of skills needed to complete the projects within their programs. An organization that uses this structure earns its revenue primarily from performing projects for other groups under contract. For example, many defense, architectural, engineering, and consulting companies use a project organizational structure. These companies often hire people specifically to work on particular projects.

A **matrix organizational structure** represents the middle ground between functional and project structures. Personnel often report both to a functional manager and one or more project managers. For example, IT personnel at many companies often split their time between two or more projects, but they report to their manager in the IT department. Project managers in matrix organizations have staff from various functional areas working on their projects, as shown in Figure 2-3. Matrix organizational structures can be strong, weak, or balanced, based on the amount of control exerted by the project managers. Problems can occur if project team members are assigned to several projects in a matrix structure and the project manager does not have adequate control of their time.

Organizational Culture

Just as an organization's structure affects its ability to manage projects, so does its culture. Organizational culture is a set of shared assumptions, values, and behaviours that characterize the functioning of an organization. It often includes elements of all four frames described previously. Organizational culture is very powerful, and many people believe the underlying causes of many companies' problems are not in the organizational structure or staff; they are in the culture. It is also important to note that the same organization can have different subcultures. The IT department may have a different organizational culture than the finance department, for example. Some organizational cultures make it easier to manage projects.

According to Stephen P. Robbins and Timothy Judge, authors of a popular textbook on organizational behaviour, there are 10 characteristics of organizational culture:

1. Member identity: The degree to which employees identify with the organization as a whole rather than with their type of job or profession. For example, project managers or team members might feel more dedicated to their company or project team than to their job or profession, or they might not have any loyalty to a particular company or

team. As you can guess, an organizational culture in which employees identify more with the whole organization are more conducive to a good project culture.

2. Group emphasis: The degree to which work activities are organized around groups or teams, rather than individuals. An organizational culture that emphasizes group work is best for managing projects.
3. People focus: The degree to which management's decisions take into account the effect of outcomes on people within the organization. A project manager might assign tasks to certain people without considering their individual needs, or the project manager might know each person very well and focus on individual needs when assigning work or making other decisions. Good project managers often balance the needs of individuals and the organization.
4. Unit integration: The degree to which units or departments within an organization are encouraged to coordinate with each other. Most project managers strive for strong unit integration to deliver a successful product, service, or result. An organizational culture with strong unit integration makes the project manager's job easier.
5. Control: The degree to which rules, policies, and direct supervision are used to oversee and control employee behavior. Experienced project managers know it is often best to balance the degree of control to get good project results.
6. Risk tolerance: The degree to which employees are encouraged to be aggressive, innovative, and risk seeking. An organizational culture with a higher risk tolerance is often best for project management because projects often involve new technologies, ideas, and processes.
7. Reward criteria: The degree to which rewards, such as promotions and salary increases, are allocated according to employee performance rather than seniority, favouritism, or other non performance factors. Project managers and their teams often perform best when rewards are based mostly on performance.
8. Conflict tolerance: The degree to which employees are encouraged to air conflicts and criticism openly. It is very important for all project stakeholders to have good communications, so it is best to work in an organization where people feel comfortable discussing differences openly.

9. Means-ends orientation: The degree to which management focuses on outcomes rather than on techniques and processes used to achieve results. An organization with a balanced approach in this area is often best for project work.
10. Open-systems focus: The degree to which the organization monitors and responds to changes in the external environment. As you learned earlier in this chapter, projects are part of a larger organizational environment, so it is best to have a strong open-systems focus.

As you can see, there is a definite relationship between organizational culture and successful project management. Project work is most successful in an organizational culture where employees identify more with the organization, where work activities emphasize groups, and where there is strong unit integration, high risk tolerance, performance-based rewards, high conflict tolerance, an open-systems focus, and a balanced focus on people, control, and means orientation.

2.4 STAKEHOLDER MANAGEMENT

Project stakeholders are the people involved in or affected by project activities. Stakeholders can be internal or external to the organization, directly involved in the project, or simply affected by the project. Internal project stakeholders include the project sponsor, project team, support staff, and internal customers of the project. Other internal stakeholders include top management, other functional managers, and other project managers. Projects affect these additional internal stakeholders because they use the organization's limited resources. Thus, while additional internal stakeholders may not be directly involved in the project, they are still stakeholders because the project affects them in some way.

External project stakeholders include the project's customers (if they are external to the organization), competitors, suppliers, and other external groups potentially involved in the project or affected by it, such as government officials or concerned citizens. Because the purpose of project management is to meet project requirements and satisfy stakeholders, it is critical that project managers take adequate time to identify, understand, and manage relationships with all project stakeholders.

THE IMPORTANCE OF TOP MANAGEMENT COMMITMENT

People in top management positions, of course, are key stakeholders in projects. A very important factor in helping project managers successfully lead projects is the level of commitment and support they receive from top management. In fact, without top management commitment, many projects will fail.

As described earlier, projects are part of the larger organizational environment, and many factors that might affect a project are out of the project manager's control. Several studies cite executive support as one of the key factors associated with the success of virtually all projects.

Top management commitment is crucial to project managers for the following reasons:

a) Project managers need adequate resources. The best way to kill a project is to withhold the required money, people, resources, and visibility for the project. If project managers have top management commitment, they will also have adequate resources and not be distracted by events that do not affect their specific projects.
b) Project managers often require approval for unique project needs in a timely manner. For example, on large information technology projects, top management must understand that unexpected problems may result from the nature of the products being produced and the specific skills of the people on the project team. For example, the team might need additional hardware and software halfway through the project for proper testing, or the project manager might need to offer special pay and benefits to attract and retain key project personnel. With top management commitment, project managers can meet these specific needs in a timely manner.
c) Project managers must have cooperation from people in other parts of the organization. Since most information technology projects cut across functional areas, top management must help project managers deal with the political issues that often arise in these types of situations. If certain functional managers are not responding to project managers' requests for necessary information, top management must step in to encourage functional managers to cooperate.
d) Project managers often need someone to mentor and coach them on leadership issues. Many information technology project managers come from technical positions and often are inexperienced as managers. Senior managers should take the time to pass on

leadership advice and encourage new project managers to take classes to develop leadership skills and allocate the time and funds for them to do so.

Information technology project managers work best in an environment in which top management values information technology. Working in an organization that values good project management and sets standards for its use also helps project managers succeed.

2.5 PROJECT PHASES

A project life cycle is a collection of phases. Phases break projects down into smaller, more manageable pieces, which will reduce uncertainty. Some organizations specify a set of life cycles for use in all of their projects, while others follow common industry practices based on the types of projects involved. Project life cycles define what work will be performed in each phase, what deliverables will be produced and when, who is involved in each phase, and how management will control and approve work produced in each phase. A **deliverable** is a product or service, such as a technical report, a training session, a piece of hardware, or a segment of software code, produced or provided as part of a project.

In early phases of a project life cycle, resource needs are usually lowest and the level of uncertainty is highest. Project stakeholders have the greatest opportunity to influence the final characteristics of the project's products, services, or results during the early phases of a project life cycle. It is much more expensive to make major changes to a project during later phases. During the middle phases of a project life cycle, the certainty of completing the project improves as it continues and as more information is known about the project requirements and objectives. Also, more resources are usually needed than during the initial or final phase. The final phase of a project focuses on ensuring that project requirements were met and that the project sponsor approves completion of the project.

The first two traditional project phases (concept and development) focus on planning, and are often referred to as **project feasibility**. The last two phases (implementation and closeout) focus on delivering the actual work, and are often referred to as **project acquisition**. Each phase of a project should be successfully completed before the team moves on to the next phase. This project life cycle approach provides better management control and appropriate links to the ongoing operations of the organization.

Figure 2.3 provides a summary of the general phases of the traditional project life cycle. In the concept phase, managers usually develop a business case, which describes the need for

the project and basic underlying concepts. A preliminary or rough cost estimate is developed in this first phase, and an overview of the required work is created.

One tool for creating an overview of the required work is a work breakdown structure (WBS). A WBS outlines project work by decomposing the work activities into different levels of tasks. The WBS is a deliverable-oriented document that defines the total scope of the project. In the concept phase, a WBS usually has only two levels.

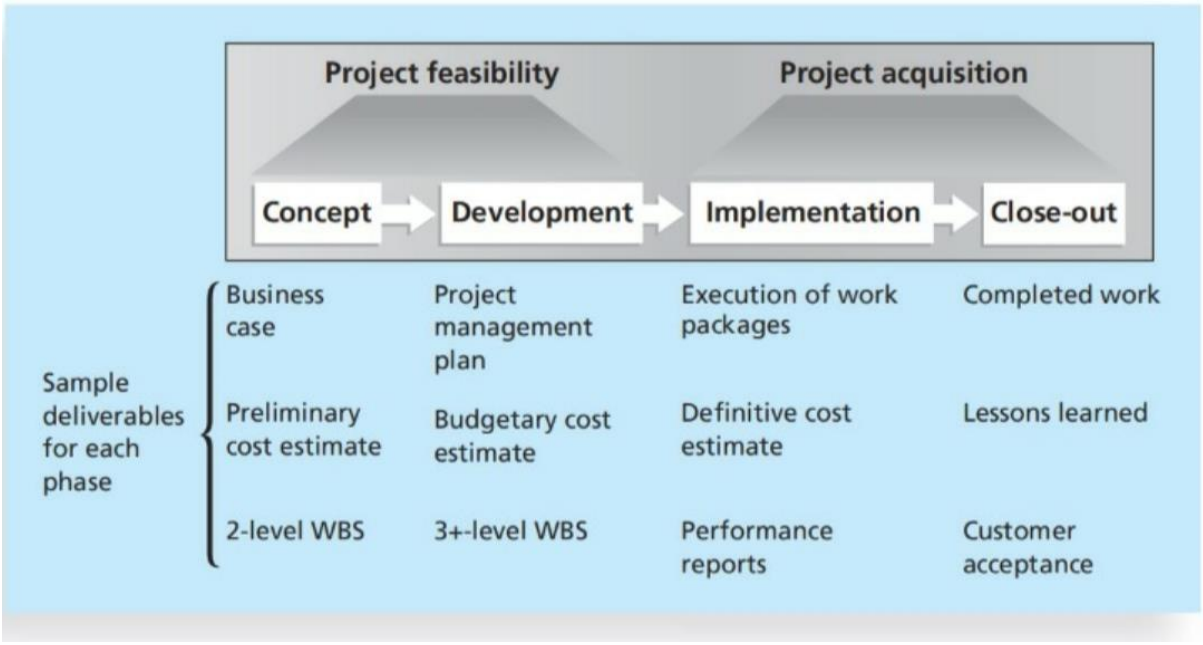


Figure 2.3 – Phases of traditional Project Life Cycle

After the concept phase is completed, the next project phase—development—begins. In the development phase, the project team creates more detailed project management plans, a more accurate cost estimate, and a more thorough WBS.

The third phase of the traditional project life cycle is implementation. In this phase, the project team creates a definitive or very accurate cost estimate, delivers the required work, and provides performance reports to stakeholders.

The last phase of the traditional project life cycle is the close-out phase. In it, all of the work is completed, and customers should accept the entire project. The project team should document its experiences on the project in a lessons-learned report.

Many projects, however, do not follow this traditional project life cycle. They still have general phases with some similar characteristics, but they are much more flexible. For example, a project might have just three phases—the initial, intermediate, and final phases. Or, there may be multiple intermediate phases. A separate project might be needed just to complete a feasibility study. Regardless of the project life cycle's specific phases, it is good

practice to think of projects as having phases that connect the beginning and end of the process. This way, people can measure progress toward achieving project goals during each phase and the project is more likely to be successful.

2.6 THE CONTEXT OF INFORMATION TECHNOLOGY PROJECTS

The project context has a critical impact on which product development life cycle will be most effective for a particular software development project. Likewise, several issues unique to the IT industry have a critical impact on managing IT projects. These include the nature of projects, the characteristics of project team members, and the diverse nature of technologies involved.

2.6.1 THE NATURE OF IT PROJECTS

Unlike projects in many other industries, IT projects are diverse. Some involve a small number of people installing off-the-shelf hardware and associated software. Others involve hundreds of people analyzing several organizations' business processes and then developing new software in a collaborative effort with users to meet business needs. Even for small hardware-oriented projects, a wide diversity of hardware types can be involved—personal computers, mainframe computers, network equipment, kiosks, laptops, tablets, or smartphones. The network equipment might be wireless, cellular based, or cable-based, or might require a satellite connection. The nature of software development projects is even more diverse than hardware-oriented projects. A software development project might include creating a simple, stand-alone Microsoft Excel or Access application or a sophisticated, global e-commerce system that uses state-of-the-art programming languages and runs on multiple platforms.

IT projects also support every possible industry and business function. Managing an IT project for a film company's animation department requires different knowledge and skills than a project to improve a federal tax collection system or to install a communication infrastructure in a third-world country. Because of the diversity of IT projects and the newness of the field, it is important to develop and follow best practices in managing these varied projects. Developing best practices gives IT project managers a common starting point and method to follow with every project.

2.6.2 CHARACTERISTICS OF IT PROJECT TEAM MEMBERS

Because IT projects are diverse, the people involved come from diverse backgrounds and possess different skills. The resulting diverse project teams provide a significant advantage because they can analyze project requirements from a more robust systems view. Many companies purposely hire graduates with degrees in other fields such as business, mathematics, or the liberal arts to provide different perspectives on IT projects. Even with these different educational backgrounds, however, there are common job titles for people working on most IT projects, such as business analyst, programmer, network specialist, database analyst, quality assurance expert, technical writer, security specialist, hardware engineer, software engineer, and system architect. Within the category of programmer, several other job titles describe the specific technologies used, such as Java programmer, PHP programmer, and C/C++/C# programmer.

Some IT projects require the skills of people in just a few job functions, but some require inputs from many or all of them. Occasionally, IT professionals move between these job functions, but more often people become technical experts in one area or they decide to move into a management position. It is also rare for technical specialists or project managers to remain with the same company for a long time. In fact, many IT projects 65include a large number of contract workers. Working with this "army of free agents," as author Rob Thomsett calls them, creates special challenges.

2.6.3 DIVERSE TECHNOLOGIES

Many of the job titles for IT professionals reflect the different technologies required to hold those positions. Differences in technical knowledge can make communication between professionals challenging. Hardware specialists might not understand the language of database analysts, and vice versa. Security specialists may have a hard time communicating with business analysts. People within the same IT job function often do not understand each other because they use different technology. For example, someone with the title of programmer can often use several different programming languages. However, if programmers are limited in their ability to work in multiple languages, project managers might find it more difficult to form and lead more versatile project teams.

Another problem with diverse technologies is that many of them change rapidly. A project team might be close to finishing a project when it discovers a new technology that can greatly enhance the project and better meet long-term business needs. New technologies have also

shortened the time frame many businesses have to develop, produce, and distribute new products and services. This fast-paced environment requires equally fast paced processes to manage and produce IT projects and products.

2.7 RECENT TRENDS AFFECTING INFORMATION TECHNOLOGY PROJECT MANAGEMENT

Recent trends such as increased globalization, outsourcing, virtual teams, and agile project management are creating additional challenges and opportunities for IT project managers and their teams. Each of these trends and suggestions for addressing them are discussed in this section.

2.7.1 GLOBALIZATION

IT is a key enabler of globalization. In 2014, more than 1.3 billion people were using Facebook, spending an average of 21 minutes a day.17 Other social networks, such as Twitter and LinkedIn, also continue to grow. In 2014, there were over 284 million Twitter users and 332 million LinkedIn users. According to LinkedIn's website, in the third quarter of 2014, 75 percent of new members came from outside the United States. Globalization has significantly affected the field of IT. Even though major IT companies such as Apple, IBM, and Microsoft started in the United States, much of their business is global—indeed, companies and individuals throughout the world contribute to the growth of information technologies, and work and collaborate on various IT projects.

It is important for project managers to address several key issues when working on global projects:

Communications: Because people work in different time zones, speak different languages, have different cultural backgrounds, and celebrate different holidays, it is important to address how people will communicate in an efficient and timely manner. A communications management plan is vital. For details, see the plan described in Chapter 10, Project Communications Management.

Trust: Trust is an important issue for all teams, especially when they are global teams. It is important to start building trust immediately by recognizing and respecting others' differences and the value they add to the project.

Common work practices*:* It is important to align work processes and develop a modus operandi with which everyone agrees and is comfortable. Project managers must allow time for the team to develop these common work practices. Using special tools, as described next, can facilitate this process.

Tools: IT plays a vital role in globalization, especially in enhancing communications and work practices. Many people use free tools such as Skype, Google Docs, or social media to communicate. Many project management software tools include their own communications and collaboration features in an integrated package. IBM continues to be the leader in providing collaboration tools to businesses in over 175 companies, followed by Oracle in 145 countries, SAP in 130 countries, and Microsoft in 113 countries.18 Work groups must investigate options and decide which tools will work best for their projects. Security is often a key factor in deciding which tools to use.

2.7.2 OUTSOURCING

The term offshoring is sometimes used to describe outsourcing from another country. Offshoring is a natural outgrowth of globalization. IT projects continue to rely more and more on outsourcing, both within and outside their country boundaries.

In its simplest sense, **Software Development Outsourcing** describes an arrangement, in which an organization chooses to hire an external software development agency to effectively carry out all the tasks of a software development project, that could be done in-house instead. Software outsourcing is about the practice of a company handing over the control of a certain business process or project to a third-party vendor that is qualified and capable of handling the required business tasks.

Outsourcing is cost-effective, and in particular - offshore software outsourcing helps lower development costs which essentially translates to lower market price and enhances competition. However, in recent years, according to a report by Deloitte and Dubai Outsource City, companies are starting to look toward software outsourcing to achieve a variety of business objectives, beyond just costs.

Modes of Software Outsourcing:

There are certain ways for businesses to outsource their software projects to vendors across the globe, where the development centres can reside on-shore, offshore, and near-shore. Let’s consider in detail as follow:

Onshore Software Outsourcing

Onshore outsourcing refers to the act of customer companies working with development teams of software companies that are located in the same country. The advantage of onshore outsourcing is that there are virtually no language barriers which make communication much easier and eventually, making outsourcing more effective. However, in return, customers may have to pay more for development costs.

Offshore Software Outsourcing

Offshore outsourcing means working with development teams in other countries. This is the most cost-effective option due to low labour costs, and also online communications channels (e.g. Emails, VoIP Phones, Zoom video conferences, etc.) making it possible to effectively manage software projects remotely.

Nearshore Software Outsourcing

Nearshore outsourcing companies work with customers in neighbouring countries.

HOW TO OUTSOURCE SOFTWARE DEVELOPMENT EFFECTIVELY?

There are numerous factors that influence the likelihood of success when it comes to software outsourcing. Among those, there are certain aspects which companies should keep in mind, including:

- Establish clear goals for outsourcing
- Involve in project management and collaborate with remote teams.
- Have realistic expectations.
- Set milestones and frequently track progress to provide feedback

Again, there's no way to guarantee that an outsourcing project will become a hundred percent success. But there are certain things to do that can help companies improve the chance of success. Furthermore, customer companies should opt for outsourcing partners who apply agile methodology in managing their development projects in a flexible way, to allow for better quality and more frequent release as well as improve collaborations. Additionally, employing advanced tools for efficient project task management is also highly recommended to help gain visibility into project progress.

2.7.3 VIRTUAL TEAMS

A **virtual team** (aka "virtual workgroup") is a group of people who participate in common projects by making collaborative efforts to achieve shared goals and objectives. These people perform tasks and jobs in a virtual work environment created and maintained through IT and software technologies.

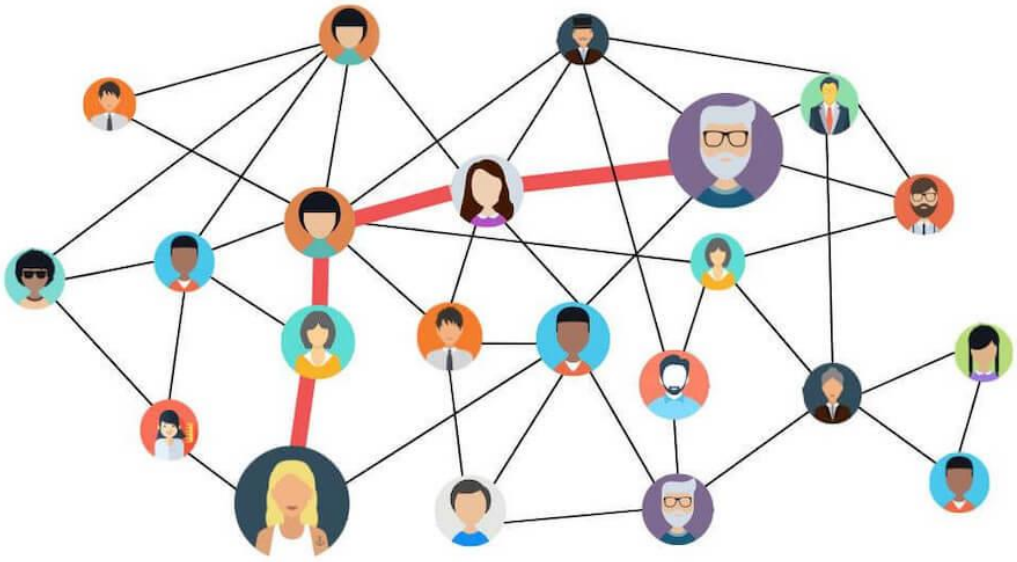


Figure 2.4 – Virtual teams in the Organization

There are two types of virtual teams, such as follows:

- **Global virtual team**. As a rule, these teams are located in different countries and cities all over the world. They can be employees of several companies which join their efforts and resources (incl. people, technology, money) to perform shared outsourced projects and achieve common goals.
- **Local virtual team**. Members of a local virtual workgroup usually belong to the same company. That company is either big or small, and it has enough resources (technology is essential) to establish and maintain virtual team workplaces and organize its employees into a productive remote group.

Virtual team management includes, but not limited to, the following processes:

- **Assembling**. Probation periods are the first measurements to be applied when starting with remote teamwork organization. The team leader should decide on those people who meet all the requirements of probation periods.

- **Training**. During this process, the team leader sets expectations as to future virtual teaming and then develops and applies a group training methodology to teach the team members how to meet the expectations.
- **Managing**. This process means using telecommunication technologies to manage ongoing tasks and jobs of remote group members.
- **Controlling**. The team leader establishes performance measures to assess and evaluate team performance. This person needs to find out whether the team is on the right track and can achieve project goals on schedule.

These are the major processes of virtual team management. However, there can be subsidiary processes that allow for a better of understanding the virtual team's phenomena.

Advantages and Dis-advantages of Virtual Team Management:

Some of the **advantages** of virtual team management are (but not limited to) the following:

- Reduced rents and technology savings
- Lower transportation costs and less time spent on commuting
- Instant communication and information exchange

Some of the **disadvantages** of virtual team management are (but not limited to) the following:

- Poorer control of virtual groups (this may result in reduced trust in virtual teams), because there are no direct control tools
- Problems to establish good virtual team leadership (comparing to "physical" team leading)
- Unfitness to the projects which require on-site control and management

How to improve virtual team collaboration

As mentioned, virtual team management offers tangible benefits to your project or business; still, you have to address some distinct challenges. Many project managers struggle to create a truly "collaborative" virtual culture.

These two essential steps below can help you build a productive virtual team and improve remote collaboration.

1. Use a single collaborative system for virtual team management

Make sure that everyone in your remote team uses the same virtual application and database to manage inbox, chats, to-do lists, shared documents, spreadsheets, and other essentials of your projects. For example, you can successfully manage your virtual team remotely from

anywhere with any device by adopting innovative cloud products such as Citrix VDI and Cloud QuickBooks Hosting from Apps4Rent.

Remote employees often spread information across multiple systems and tools. For example, using Gmail for emailing, Zoho for CRM, Slack for real-time communication, and Skype for video conferencing can be okay to some extent.

However, as your projects grow and you onboard new remote workers, your team has to deal with multiple data sources and various apps, which will ultimately create clutter and lack of control of team performance.

2. Enable automation to get more things done faster

As artificial intelligence, IoT, edge computing, and other emerging technologies become increasingly prevalent; project managers and team leaders seek ways to improve process management and collaboration in virtual team environments.

Automation is among the simplest ways to help remote teams get rid of routine tasks and focus on more value-adding activities. It allows for maximizing ROI on human capital (including virtual teams), while reducing time and effort needed to get things done.

For example, let's take online sales and lead generation. Here are at least three things that workflow automation can offer for remote sales teams and marketers:

1. Simplify lead data capturing by enabling web forms and pop-up invitations as well as automated follow-up emails and smart notifications for customers. Examples: Insightly CRM, Zoho, Pipedrive.

2. Shorten total lead time by as much as 35% by allowing for chatbots and intelligent virtual assistants to serve online prospects and customers. Examples: livechatinc.com, Zendesk AI-powered chatbot, chatbot.com.

3. Delight customers by gamifying lead generation. The secret here is, instead of wasting time on cold messages via phone, social media, or email, your sales reps will focus on analysing customer data. Your team gets insights about what a given prospect is seeking and what product or service is best to offer. Quizzes, video guides, and interactive HTML5 mini-games are examples of how you can gamify lead generation and help your local and remote teams get more deals.

2.8 CHECK YOUR PROGRESS

1. What are Systems approach, Systems philosophy and Systems analysis in Project Management?
2. Which are the Four Frames of Organizations?
3. Name any 5 characteristics of organizational culture.
4. What are the key issues faced by Project Managers while working on global projects?
5. Which are the different modes of outsourcing?

Answers to check your progress:

1. Systems approach is a holistic and analytical approach to solving complex problems that includes using a systems philosophy, systems analysis, and systems management. Systems philosophy is an overall model for thinking about things as systems. Systems analysis is a problem-solving approach that requires defining the scope of the system, dividing it into components, and then identifying and evaluating its problems, opportunities, constraints, and needs.
2. Structural frame, human resources (HR) frame, political frame, symbolic frame.
3. Member identity, Group emphasis, People focus, Unit integration, Control
4. Communications, Trust, Common work practices, Tools
5. Onshore Software Outsourcing, Offshore Software Outsourcing, Nearshore Software Outsourcing.

2.9 SUMMARY

This unit focuses on the systems view of project management and the application of Information Technology. The analysis helps the organization to provide the perspectives of structural, hr and political frames. This also helps to understand the conceptual, developmental, implementational and close-out phases of project life cycle. It also provides the clarity between project and product development.

2.10 KEYWORDS

- **Systems philosophy:** It is an overall model for thinking about things as systems.
- **Systems analysis:** It is a problem-solving approach that requires defining the scope of the system, dividing it into components, and then identifying and evaluating its problems, opportunities, constraints, and needs.

- **Political frame:** It addresses organizational and personal politics.
- **Matrix organizational structure:** It represents the middle ground between functional and project structures.
- **Deliverable:** It is a product or service, such as a technical report, a training session, a piece of hardware, or a segment of software code, produced or provided as part of a project.

2.11 QUESTIONS FOR SELF STUDY

1. Explain the four frames of organizations and their perspectives.
2. Explain Functional, project and matrix organizational structures with the help of a diagram.
3. What are the characteristics of organizational culture according to Stephen P. Robbins and Timothy Judge?
4. Why top management commitment is crucial to project managers?
5. Explain phases of traditional project life cycle with the help of a neat diagram.
6. What the key issues unique to IT industry that has a critical impact on managing IT projects?
7. Write a brief note on recent trends affecting information technology project management.

2.12 REFERENCES

1. https://ocw.ui.ac.id/pluginfile.php/13396/mod_resource/content/2/Schwalbe%208th%20-%20Chapter%2002.pdf
2. https://www.tpptechnology.com/
3. https://mymanagementguide.com/

UNIT-3: SOFTWARE MANAGEMENT CONCEPTS

STRUCTURE

3.0 OBJECTIVES

After studying this unit, you will be able to:

- Explain the major focusing factors of the management, 4Ps – People, Process, Product, Project

- Identify different kinds of people involved which are stake holders, team leaders, software team, etc.
- Describe the scope of the project and the ability to decompose the problem
- Explain the SCM scenario with the example of WebApps
- Distinguish the different types of Software Testing Strategies and the implementation need
- Discuss the debugging strategy and clean room strategy

3.1 INTRODUCTION

In this unit, we are going to discuss about software management concepts. The Management Spectrum focuses on the four P's: people, product, process, and project. It emphasizes on the importance of these 4 parameters in building a successful and robust team and process in software project management.

3.2 THE MANAGEMENT SPECTRUM

Effective software project management focuses on the four P's: people, product, process, and project. The order is not arbitrary. The manager who forgets that software engineering work is an intensely human endeavor will never have success in project management. A manager who fails to encourage comprehensive stakeholder communication early in the evolution of a product risks building an elegant solution for the wrong problem. The manager who pays little attention to the process runs the risk of inserting competent technical methods and tools into a vacuum. The manager who embarks without a solid project plan jeopardizes the success of the project.

3.2.1 THE PEOPLE

According to People Capability Maturity Model (pcmm), the people factor defines the key factor areas in software development**:** staffing, communication and coordination, work environment, performance management, training, compensation, competency analysis and development, career development, workgroup development, team/culture development, and others. Organizations that achieve high levels of People-CMM maturity have a higher likelihood of implementing effective software project management practices.

3.2.1.1 THE STAKEHOLDERS

The software process (and every software project) is populated by stakeholders who can be categorized into one of five constituencies:

1. Senior managers who define the business issues that often have a significant influence on the project.
2. Project (technical) managers who must plan, motivate, organize, and control the practitioners who do software work.
3. Practitioners who deliver the technical skills that are necessary to engineer a product or application.
4. Customers who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. End users who interact with the software once it is released for production use.

Every software project is populated by people who fall within this taxonomy. To be effective, the project team must be organized in a way that maximizes each person's skills and abilities. And that's the job of the team leader.

3.2.1.2 TEAM LEADERS

Project management is a people-intensive activity, and for this reason, competent practitioners often make poor team leaders. They simply don't have the right mix of people skills. According to Jerry Weinberg, the expected skills to be present in team leaders are,

- **Motivation**. The ability to encourage (by "push or pull") technical people to produce to their best ability.
- **Organization**. The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
- **Ideas or innovation**. The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.
- **Problem solving**. An effective software project manager can diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution, apply lessons learned from past projects to new situations, and remain flexible enough to change direction if initial attempts at problem solution are fruitless.

- **Managerial identity.** A good project manager must take charge of the project. She must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.
- **Achievement**. A competent manager must reward initiative and accomplishment to optimize the productivity of a project team. She must demonstrate through her own actions that controlled risk taking will not be punished.
- **Influence and team building**. An effective project manager must be able to "read" people; she must be able to understand verbal and nonverbal signals and react to the needs of the people sending these signals. The manager must remain under control in high-stress situations.

3.2.1.3 THE SOFTWARE TEAM

The "best" team structure depends on the management style of the organization, the number of people who will populate the team and their skill levels, and the overall problem difficulty. Mantei [Man81] describes seven project factors that should be considered when planning the structure of software engineering teams:

- Difficulty of the problem to be solved
- "Size" of the resultant program(s) in lines of code or function points T
- ime that the team will stay together (team lifetime)
- Degree to which the problem can be modularized
- Required quality and reliability of the system to be built
- Rigidity of the delivery date
- Degree of sociability (communication) required for the project

3.2.2 THE PRODUCT

A software project manager is confronted with a dilemma at the very beginning of a software project. Quantitative estimates and an organized plan are required, but solid information is unavailable. A detailed analysis of software requirements would provide necessary information for estimates, but analysis often takes weeks or even months to complete. Worse, requirements may be fluid, changing regularly as the project proceeds. Yet, a plan is needed "now!" Like it or not, one must examine the product and the problem it is intended to solve at the very beginning of the project. At a minimum, the scope of the product must be established and bounded.

3.2.2.1 SOFTWARE SCOPE

The first software project management activity is the determination of software scope. Scope is defined by answering the following questions:

Context. How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?

Information objectives. What customer-visible data objects are produced as output from the software? What data objects are required for input? Function and performance. What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Software project scope must be unambiguous and understandable at the management and technical levels. A statement of software scope must be bounded. That is, quantitative data (e.g., number of simultaneous users, target environment, maximum allowable response time) are stated explicitly, constraints and/or limitations (e.g., product cost restricts memory size) are noted, and mitigating factors (e.g., desired algorithms are well understood and available in Java) are described.

3.2.2.2 PROBLEM DECOMPOSITION

Problem decomposition, sometimes called partitioning or problem elaboration, is an activity that sits at the core of software requirements analysis. During the scoping activity no attempt is made to fully decompose the problem. Rather, decomposition is applied in two major areas:

(1) The functionality and content (information) that must be delivered and

(2) The process that will be used to deliver it.

Human beings tend to apply a divide-and-conquer strategy when they are confronted with a complex problem. Stated simply, a complex problem is partitioned into smaller problems that are more manageable. This is the strategy that applies as project planning begins. Software functions, described in the statement of scope, are evaluated and refined to provide more detail prior to the beginning of estimation.

Example:

As an example, consider a project that will build a new word-processing product. Among the unique features of the product are continuous voice as well as virtual keyboard input via a multi-touch screen, extremely sophisticated “automatic copy edit” features, page layout

capability, automatic indexing and table of contents, and others. The project manager must first establish a statement of scope that bounds these features (as well as other more mundane functions such as editing, file management, and document production). For example, will continuous voice input require that the product be "trained" by the user? Specifically, what capabilities will the copy edit feature provide? Just how sophisticated will the page layout capability be and will it encompass the capabilities implied by a multitouch screen?

As the statement of scope evolves, a first level of partitioning naturally occurs. The project team learns that the marketing department has talked with potential customers and found that the following functions should be part of automatic copy editing:

(1) Spell checking,
(2) Sentence grammar checking,
(3) Reference checking for large documents (e.g., Is a reference to a bibliography entry found in the list of entries in the bibliography?),
(4) The implementation of a style sheet feature that imposed consistency across a document, and
(5) Section and chapter reference validation for large documents.

Each of these features represents a sub-function to be implemented in software. Each can be further refined if the decomposition will make planning easier.

3.2.3 THE PROCESS

The purpose of the process element is, the team must decide which process model is most appropriate for

(1) The customers who have requested the product and the people who will do the work,
(2) The characteristics of the product itself, and
(3) The project environment in which the software team works.

When a process model has been selected, the team then defines a preliminary project plan based on the set of process framework activities. Once the preliminary plan is established, process decomposition begins. That is, a complete plan, reflecting the work tasks required to populate the framework activities must be created.

3.2.3.1 PROCESS DECOMPOSITION

A software team should have a significant degree of flexibility in choosing the software process model that is best for the project and the software engineering tasks that populate the process model once it is chosen. Once the process model has been chosen, the process framework is adapted to it.

Process decomposition commences when the project manager asks, "How do we accomplish this framework activity?" For example, a small, relatively simple project might require the following work tasks for the communication activity:

1. Develop list of clarification issues.
2. Meet with stakeholders to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required.

These events might occur over a period of less than 48 hours. They represent a process decomposition that is appropriate for the small, relatively simple project.

3.2.4 THE PROJECT

In order to manage a successful software project, one has to understand what can go wrong so that problems can be avoided. In an excellent paper on software projects, John Reel [Ree99] defines 10 signs that indicate that an information systems project is in jeopardy:

1. Software people don't understand their customer's needs.
2. The product scope is poorly defined.
3. Changes are managed poorly.
4. The chosen technology changes.
5. Business needs change [or are ill defined].
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost [or was never properly obtained].
9. The project team lacks people with appropriate skills.
10. Managers [and practitioners] avoid best practices and lessons learned.

Reel [Ree99] suggests a five-part common sense approach to software projects:

1. *Start on the right foot.* This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations for everyone who will be involved in the project. It is reinforced by building the right team and giving the team the autonomy, authority, and technology needed to do the job.
2. *Maintain momentum.* Many projects get off to a good start and then slowly disintegrate. To maintain momentum, the project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.
3. *Track progress.* For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using technical reviews) as part of a quality assurance activity. In addition, software process and project measures can be collected and used to assess progress against averages developed for the software development organization.
4. *Make smart decisions*. In essence, the decisions of the project manager and the software team should be to "keep it simple." Whenever possible, decide to use commercial off-the-shelf software or existing software components or patterns, decide to avoid custom interfaces when standard approaches are available, decide to identify and then avoid obvious risks, and decide to allocate more time than you think is needed to complex or risky tasks (you'll need every minute).
5. *Conduct a post-mortem analysis.* Establish a consistent mechanism for extracting lessons learned for each project. Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from team members and customers, and record findings in written form.

3.3 SOFTWARE CONFIGURATION MANAGEMENT

The output of the software process is information that may be divided into three broad categories:

(1) Computer programs (both source level and executable forms),

(2) Work products that describe the computer programs (targeted at various stakeholders), and

(3) Data or content (contained within the program or external to it).

The items that comprise all information produced as part of the software process are collectively called a software configuration.

3.3.1 SCM SCENARIO

A typical CM operational scenario involves a project manager who is in charge of a software group, a configuration manager who is in charge of the CM procedures and policies, the software engineers who are responsible for developing and maintaining the software product, and the customer who uses the product. In the scenario, assume that the product is a small one involving about 15,000 lines of code being developed by a team of six people. (Note that other scenarios of smaller or larger teams are possible, but, in essence, there are generic issues that each of these projects face concerning CM.)

At the operational level, the scenario involves various roles and tasks. For the project manager, the goal is to ensure that the product is developed within a certain time frame. Hence, the manager monitors the progress of development and recognizes and reacts to problems. This is done by generating and analyzing reports about the status of the software system and by performing reviews on the system

The goals of the configuration manager are to ensure that procedures and policies for creating, changing, and testing of code are followed, as well as to make information about the project accessible. To implement techniques for maintaining control over code changes, this manager introduces mechanisms for making official requests for changes, for evaluating them (via a Change Control Board that is responsible for approving changes to the software system), and for authorizing changes. The manager creates and disseminates task lists for the engineers and basically creates the project context. Also, the manager collects statistics about components in the software system, such as information determining which components in the system are problematic.

For the software engineers, the goal is to work effectively. This means engineers do not unnecessarily interfere with each other in the creation and testing of code and in the production of supporting work products. But, at the same time, they try to communicate and coordinate efficiently. Specifically, engineers use tools that help build a consistent software product. They communicate and coordinate by notifying one another about tasks required and tasks completed. Changes are propagated across each other’s work by merging files. Mechanisms exist to ensure that, for components that undergo simultaneous changes, there is some way of resolving conflicts and merging changes. A history is kept of the evolution of all

components of the system along with a log with reasons for changes and a record of what actually changed. The engineers have their own workspace for creating, changing, testing, and integrating code. At a certain point, the code is made into a baseline from which further development continues and from which variants for other target machines are made.

The customer uses the product. Since the product is under CM control, the customer follows formal procedures for requesting changes and for indicating bugs in the product.

Ideally, a CM system used in this scenario should support all these roles and tasks; that is, the roles determine the functionality required of a CM system. The project manager sees CM as an auditing mechanism; the configuration manager sees it as a controlling, tracking, and policy making mechanism; the software engineer sees it as a changing, building, and access control mechanism; and the customer sees it as a quality assurance mechanism.

SCM ELEMENTS

In her comprehensive white paper on software configuration management, Susan Dart [Dar01] identifies four important elements that should exist when a configuration management system is developed:

- **Component elements**—a set of tools coupled within a file management system (e.g., a database) that enables access to and management of each software configuration item.
- **Process elements**—a collection of actions and tasks that define an effective approach to change management (and related activities) for all constituencies involved in the management, engineering, and use of computer software.
- **Construction elements**—a set of tools that automate the construction of software by ensuring that the proper set of validated components (i.e., the correct version) have been assembled.
- **Human elements**—a set of tools and process features (encompassing other CM elements) used by the software team to implement effective SCM. These elements (to be discussed in more detail in later sections) are not mutually exclusive.

For example, component elements work in conjunction with construction elements as the software process evolves. Process elements guide many human activities that are related to SCM and might therefore be considered human elements as well.

3.3.2 SCM PROCESS

The software configuration management process defines a series of tasks that have four primary objectives:

1. To identify all items that collectively define the software configuration,
2. To manage changes to one or more of these items,
3. To facilitate the construction of different versions of an application, and
4. To ensure that software quality is maintained as the configuration evolves over time.

A process that achieves these objectives need not be bureaucratic or ponderous, but it must be characterized in a manner that enables a software team to develop answers to a set of complex questions:

- How does a software team identify the discrete elements of a software configuration?
- How does an organization manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
- How does an organization control changes before and after software is released to a customer?
- Who has responsibility for approving and ranking requested changes?
- How can we ensure that changes have been made properly?
- What mechanism is used to apprise others of changes that are made?

These questions lead to the definition of five SCM tasks—identification, version control, change control, configuration auditing, and reporting—illustrated in Figure below.

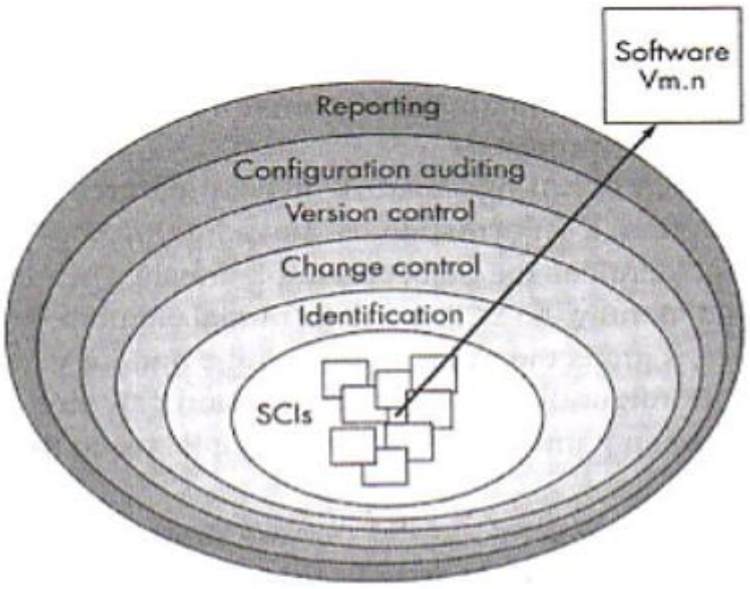


Fig 3.1: Layers of SCM process

Referring to the figure, SCM tasks can viewed as concentric layers. SCIs flow outward through these layers throughout their useful life, ultimately becoming part of the software configuration of one or more versions of an application or system. As an SCI moves through a layer, the actions implied by each SCM task may or may not be applicable. For example, when a new SCI is created, it must be identified. However, if no changes are requested for the SCI, the change control layer does not apply. The SCI is assigned to a specific version of the software (version control mechanisms come into play). A record of the SCI (its name, creation date, version designation, etc.) is maintained for configuration auditing purposes and reported to those with a need to know. In the sections that follow, we examine each of these SCM process layers in more detail.

3.3.3 CONFIGURATION MANAGEMENT FOR WEBAPPS

Content. A typical WebApp contains a vast array of content—text, graphics, applets, scripts, audio/video files, forms, active page elements, tables, streaming data, and many others. The challenge is to organize this sea of content into a rational set of configuration objects and then establish appropriate configuration control mechanisms for these objects. One approach is to model the WebApp content using conventional data modeling techniques attaching a set of specialized properties to each object. The static/dynamic nature of each object and its projected longevity (e.g., temporary, fixed existence, or permanent object) are examples of properties that are required to establish an effective SCM approach. For example, if a content item is changed hourly, it has temporary longevity. The control mechanisms for this item would be different (less formal) than those applied for a forms component that is a permanent object.

People. Because a significant percentage of WebApp development continues to be conducted in an ad hoc manner, any person involved in the WebApp can (and often does) create content. Many content creators have no software engineering background and are completely unaware of the need for configuration management. As a consequence, the application grows and changes in an uncontrolled fashion.

Scalability. The techniques and controls applied to a small WebApp do not scale upward well. It is not uncommon for a simple WebApp to grow significantly as interconnections with existing information systems, databases, data warehouses, and portal gateways are implemented. As size and complexity grow, small changes can have far-reaching and

unintended effects that can be problematic. Therefore, the rigor of configuration control mechanisms should be directly proportional to application scale.

3.3.3.1 WEBAPP CONFIGURATION OBJECTS

WebApps encompass a broad range of configuration objects—content objects (e.g., text, graphics, images, video, audio), functional components (e.g., scripts, applets), and interface objects (e.g., COM or CORBA). WebApp objects can be identified (assigned file names) in any manner that is appropriate for the organization. However, the following conventions are recommended to ensure that cross-platform compatibility is maintained: filenames should be limited to 32 characters in length, mixedcase or all-caps names should be avoided, and the use of underscores in file names should be avoided. In addition, URL references (links) within a configuration object should always use relative paths (e.g., ../products/alarmsensors.html).

All WebApp content has format and structure. Internal file formats are dictated by the computing environment in which the content is stored. However, rendering format (often called display format) is defined by the aesthetic style and design rules established for the WebApp. Content structure defines a content architecture; that is, it defines the way in which content objects are assembled to present meaningful information to an end user.

3.3.3.2 CONTENT MANAGEMENT

Content management is related to configuration management in the sense that a content management system (CMS) establishes a process (supported by appropriate tools) that acquires existing content (from a broad array of WebApp configuration objects), structures it in a way that enables it to be presented to an end user, and then provides it to the client-side environment for display.

The most common use of a content management system occurs when a dynamic WebApp is built. Dynamic WebApps create Web pages “on-the-fly.” That is, the user typically queries the WebApp requesting specific information. The WebApp queries a database, formats the information accordingly, and presents it to the user. For example, a music company provides a library of CDs for sale. When a user requests a CD or its e-music equivalent, a database is queried and a variety of information about the artist, the CD (e.g., its cover image or graphics), the musical content, and sample audio are all downloaded and configured into a

standard content template. The resultant Web page is built on the server side and passed to the client-side browser for examination by the end user

In the most general sense, a CMS "configures" content for the end user by invoking three integrated subsystems: a collection subsystem, a management subsystem, and a publishing subsystem.

3.3.3.3 CHANGE MANAGEMENT

To implement effective change management within the "code and go" philosophy that continues to dominate WebApp development, the conventional change control process must be modified. Each change should be categorized into one of four classes:

Class 1— a content or function change that corrects an error or enhances local content or functionality

Class 2—a content or function change that has an impact on other content objects or functional components

Class 3—a content or function change that has a broad impact across a WebApp (e.g., major extension of functionality, significant enhancement or reduction in content, major required changes in navigation)

Class 4—a major design change (e.g., a change in interface design or navigation approach) that will be immediately noticeable to one or more categories of user

Once the requested change has been categorized, it can be processed according to the algorithm set.

3.3.3.4 VERSION CONTROL

In an uncontrolled site where multiple authors have access to edit and contribute, the potential for conflict and problems arises—more so when these authors work from different offices at different times of day and night. One may spend the day improving the file index.html for a customer. After changes are done, another developer who works at home after hours, or in another office, may spend the night uploading their own newly revised version of the file index.html, completely overwriting previous work with no way to get it back! To avoid it, a version control process is required…

1. A central repository for the WebApp project should be established. The repository will hold current versions of all WebApp configuration objects (content, functional components, and others).

2. Each Web engineer creates his or her own working folder. The folder contains those objects that are being created or changed at any given time
3. The clocks on all developer workstations should be synchronized. This is done to avoid overwriting conflicts when two developers make updates that are very close to one another in time.
4. As new configuration objects are developed or existing objects are changed, they are imported into the central repository. The version control tool (see discussion of CVS in the sidebar) will manage all check-in and check-out functions from the working folders of each WebApp developer. The tool will also provide automatic e-mail updates to all interested parties when changes to the repository are made.
5. As objects are imported or exported from the repository, an automatic, timestamped log message is made. This provides useful information for auditing and can become part of an effective reporting scheme.

The version control tool maintains different versions of the WebApp and can revert to an older version if required.

3.3.3.5 AUDITING AND REPORTING

In the interest of agility, the auditing and reporting functions are deemphasized in Web engineering work. However, they are not eliminated altogether. All objects that are checked into or out of the repository are recorded in a log that can be reviewed at any point in time. A complete log report can be created so that all members of the WebApp team have a chronology of changes over a defined period of time. In addition, an automated e-mail notification (addressed to those developers and stakeholders who have interest) can be sent every time an object is checked in or out of the repository.

3.4 SOFTWARE TESTING STRATEGIES

3.4.1 A Strategic Approach to Software Testing

Testing is a set of activities that can be planned in advance and conducted systematically. For this reason a template for software testing—a set of steps into which you can place specific test case design techniques and testing methods—should be defined for the software process.

A number of software testing strategies have been proposed in the literature. A template is provided and all have the following generic characteristics:

- To perform effective testing, one should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences.
- Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. A strategy should provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise, progress must be measurable and problems should surface as early as possible.

3.4.1.1 VERIFICATION AND VALIDATION

Verification refers to the set of tasks that ensure that software correctly implements a specific function. Validation refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Verification and validation includes a wide array of SQA activities: technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility study, documentation review, database review, algorithm analysis, development testing, usability testing, qualification testing, acceptance testing, and installation testing. Although testing plays an extremely important role in V&V, many other activities are also necessary.

3.4.1.2 SOFTWARE TESTING STRATEGY – THE BIG PICTURE

The software process may be viewed as the spiral illustrated in Figure 3.2. Initially, system engineering defines the role of software and leads to software requirements analysis, where the information domain, function, behaviour, performance, constraints, and validation criteria for software are established. Moving inward along the spiral, you come to design and finally

to coding. To develop computer software, you spiral inward (counter clockwise) along streamlines that decrease the level of abstraction on each turn.

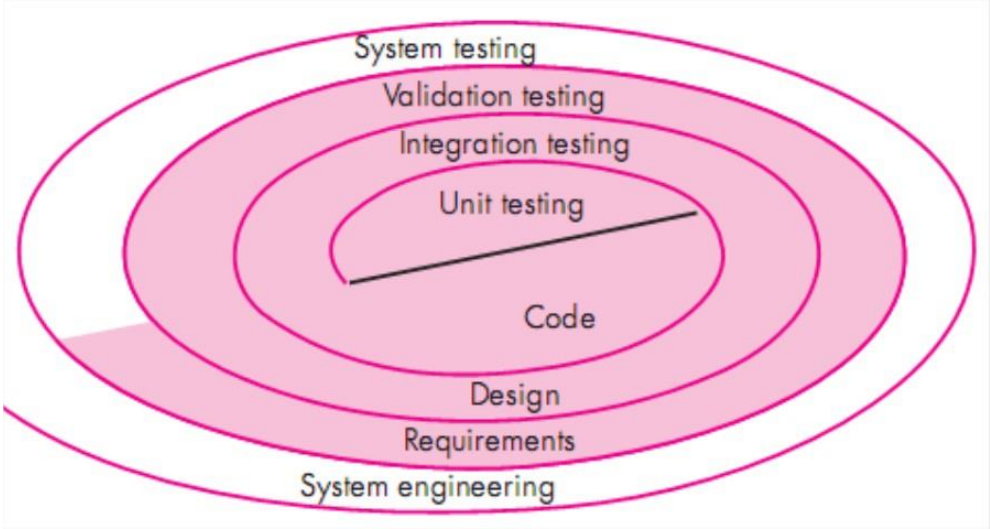


Figure – 3.2 – Software Testing Strategy

A strategy for software testing may also be viewed in the context of the spiral (Figure 3.2). Unit testing begins at the vortex of the spiral and concentrates on each unit (e.g., component, class, or WebApp content object) of the software as implemented in source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, you encounter validation testing, where requirements established as part of requirements modeling are validated against the software that has been constructed. Finally, arrives system testing, where the software and other system elements are tested as a whole. To test computer software, you spiral out in a clockwise direction along streamlines that broaden the scope of testing with each turn.

Considering the process from a procedural point of view, testing within the context of software engineering is actually a series of four steps that are implemented sequentially.

The steps are shown in Figure 3.3. Initially, tests focus on each component individually, ensuring that it functions properly as a unit. Hence, the name unit testing. Unit testing makes heavy use of testing techniques that exercise specific paths in a component's control structure to ensure complete coverage and maximum error detection. Next, components must be assembled or integrated to form the complete software package. Integration testing addresses the issues associated with the dual problems of verification and program construction. Test case design techniques that focus on inputs and outputs are more prevalent during integration,

although techniques that exercise specific program paths may be used to ensure coverage of major control paths. After the software has been integrated (constructed), a set of high-order tests is conducted. Validation criteria (established during requirements analysis) must be evaluated. Validation testing provides final assurance that software meets all informational, functional, behavioural, and performance requirements.

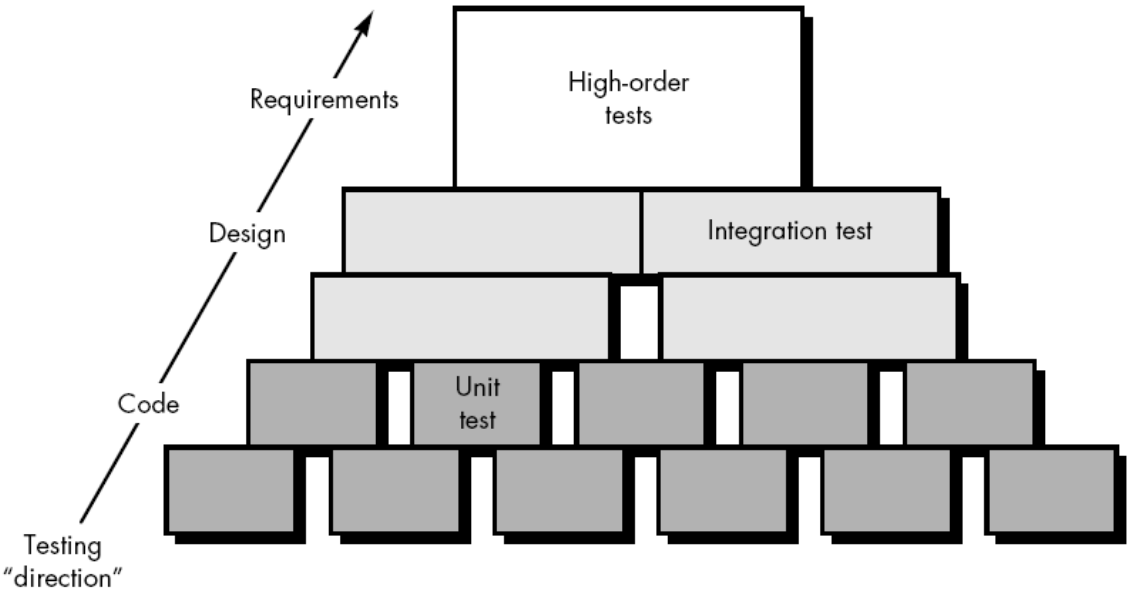


Figure 3.3 – Software Testing Steps

The last high-order testing step falls outside the boundary of software engineering and into the broader context of computer system engineering. Software, once validated, must be combined with other system elements (e.g., hardware, people, databases). System testing verifies that all elements mesh properly and that overall system function/performance is achieved.

3.4.2 STRATEGIC ISSUES

According to Tom Gilb a software testing strategy will succeed when software testers:

Specify product requirements in a quantifiable manner long before testing commences. Although the overriding objective of testing is to find errors, a good testing strategy also assesses other quality characteristics such as portability, maintainability, and usability. These should be specified in a way that is measurable so that testing results are unambiguous.

State testing objectives explicitly. The specific objectives of testing should be stated in measurable terms. For example, test effectiveness, test coverage, meantime-to-failure, the

cost to find and fix defects, remaining defect density or frequency of occurrence, and test work-hours should be stated within the test plan.

Understand the users of the software and develop a profile for each user category. Use cases that describe the interaction scenario for each class of user can reduce overall testing effort by focusing testing on actual use of the product.

Develop a testing plan that emphasizes "rapid cycle testing." Gilb recommends that a software team "learn to test in rapid cycles (2 percent of project effort) of customer-useful, at least field 'trialable,' increments of functionality and/or quality improvement." The feedback generated from these rapid cycle tests can be used to control quality levels and the corresponding test strategies.

Build "robust" software that is designed to test itself. Software should be designed in a manner that uses antibugging techniques. That is, software should be capable of diagnosing certain classes of errors. In addition, the design should accommodate automated testing and regression testing.

Use effective technical reviews as a filter prior to testing. Technical reviews can be as effective as testing in uncovering errors. For this reason, reviews can reduce the amount of testing effort that is required to produce high quality software.

Conduct technical reviews to assess the test strategy and test cases themselves. Technical reviews can uncover inconsistencies, omissions, and outright errors in the testing approach. This saves time and also improves product quality.

Develop a continuous improvement approach for the testing process. The test strategy should be measured. The metrics collected during testing should be used as part of a statistical process control approach for software testing.

3.4.3 TEST STRATEGIES FOR CONVENTIONAL SOFTWARE

A testing strategy that is chosen by most software teams falls between the two extremes. It takes an incremental view of testing, beginning with the testing of individual program units, moving to tests designed to facilitate the integration of the units, and culminating with tests that exercise the constructed system. Each of these classes of tests is described in the sections that follow.

UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors those tests uncover is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.

INTEGRATION TESTING

Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit-tested components and build a program structure that has been dictated by design.

There is often a tendency to attempt non incremental integration; that is, to construct the program using a "big bang" approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results! A set of errors is encountered. Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

Incremental integration is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied.

3.4.4 VALIDATION TESTING

Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package, and interfacing errors have been uncovered and corrected. At the validation or system level, the distinction between conventional software, object-oriented software, and WebApps disappears. Testing focuses on user-visible actions and user-recognizable output from the system.

Validation can be defined in many ways, but a simple (albeit harsh) definition is that validation succeeds when software functions in a manner that can be reasonably expected by

the customer. At this point a battle-hardened software developer might protest: “Who or what is the arbiter of reasonable expectations?” If a Software Requirements Specification has been developed, it describes all user-visible attributes of the software and contains a Validation Criteria section that forms the basis for a validation-testing approach.

After each validation test case has been conducted, one of two possible conditions exists:

1. The function or performance characteristic conforms to specification and is accepted or
2. A deviation from specification is uncovered and a deficiency list is created.

Deviations or errors discovered at this stage in a project can rarely be corrected prior to scheduled delivery. It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

3.4.5 SYSTEM TESTING

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions. In the sections that follow, I discuss the types of system tests that are worthwhile for software-based systems.

3.4.5.1 RECOVERY TESTING

Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic (performed by the system itself), re-initialization, check-pointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits

3.4.5.2 SECURITY TESTING

Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. During security testing, the tester plays the role(s) of the individual who desires to penetrate the system. Anything goes! The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to break down any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to

penetrate during recovery; may browse through insecure data, hoping to find the key to system entry.

Given enough time and resources, good security testing will ultimately penetrate a system. The role of the system designer is to make penetration cost more than the value of the information that will be obtained.

3.4.5.3 STRESS TESTING

Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example, (1) special tests may be designed that generate ten interrupts per second, when one or two is the average rate, (2) input data rates may be increased by an order of magnitude to determine how input functions will respond, (3) test cases that require maximum memory or other resources are executed, (4) test cases that may cause thrashing in a virtual operating system are designed, (5) test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.

A variation of stress testing is a technique called *sensitivity testing*. In some situations (the most common occur in mathematical algorithms), a very small range of data contained within the bounds of valid data for a program may cause extreme and even erroneous processing or profound performance degradation. Sensitivity testing attempts to uncover data combinations within valid input classes that may cause instability or improper processing.

3.4.5.4 PERFORMANCE TESTING

Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as tests are conducted. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.

Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation. That is, it is often necessary to measure resource utilization (e.g., processor cycles) in an exacting fashion. External instrumentation can monitor execution intervals, log events (e.g., interrupts) as they occur, and sample machine states on a regular basis. By instrumenting a system, the tester can uncover situations that lead to degradation and possible system failure.

3.4.5.5 DEPLOYMENT TESTING

In many cases, software must execute on a variety of platforms and under more than one operating system environment. *Deployment testing*, sometimes called *configuration testing*, exercises the software in each environment in which it is to operate. In addition, deployment testing examines all installation procedures and specialized installation software (e.g., "installers") that will be used by customers, and all documentation that will be used to introduce the software to end users.

As an example, consider the Internet-accessible version of SafeHome software that would allow a customer to monitor the security system from remote locations. The SafeHome WebApp must be tested using all Web browsers that are likely to be encountered. A more thorough deployment test might encompass combinations of Web browsers with various operating systems (e.g., Linux, Mac OS, Windows). Because security is a major issue, a complete set of security tests would be integrated with the deployment test.

3.4.6 THE ART OF DEBUGGING

Debugging is not testing but often occurs as a consequence of testing. Referring to Figure 3.4, the debugging process begins with the execution of a test case. Results are assessed and a lack of correspondence between expected and actual performance is encountered. In many cases, the noncorresponding data are a symptom of an underlying cause as yet hidden. The debugging process attempts to match symptom with cause, thereby leading to error correction.

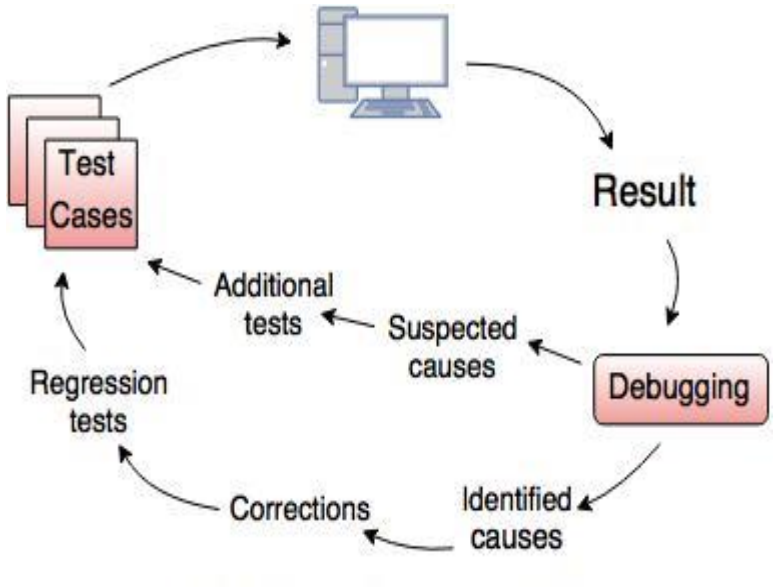


Figure 3.4 – Debugging Process

Below given are some of the characteristics of the bugs that would provide some clues:

1. The symptom and the cause may be geographically remote. That is, the symptom may appear in one part of a program, while the cause may actually be located at a site that is far removed. Highly coupled components exacerbate this situation.
2. The symptom may disappear (temporarily) when another error is corrected.
3. The symptom may actually be caused by non-errors (e.g., round-off inaccuracies).
4. The symptom may be caused by human error that is not easily traced
5. The symptom may be a result of timing problems, rather than processing problems.
6. It may be difficult to accurately reproduce input conditions (e.g., a real-time application in which input ordering is indeterminate).
7. The symptom may be intermittent. This is particularly common in embedded systems that couple hardware and software inextricably.
8. The symptom may be due to causes that are distributed across a number of tasks running on different processors

3.4.6.1 CORRECTING THE ERROR

Once a bug has been found, it must be corrected. But, as we have already noted, the correction of a bug can introduce other errors and therefore do more harm than good. Van Vleck suggests three simple questions that you should ask before making the "correction" that removes the cause of a bug:

- Is the cause of the bug reproduced in another part of the program? In many situations, a program defect is caused by an erroneous pattern of logic that may be reproduced elsewhere. Explicit consideration of the logical pattern may result in the discovery of other errors.
- What "next bug" might be introduced by the fix I'm about to make? Before the correction is made, the source code (or, better, the design) should be evaluated to assess coupling of logic and data structures. If the correction is to be made in a highly coupled section of the program, special care must be taken when any change is made.
- What could we have done to prevent this bug in the first place? This question is the first step toward establishing a statistical software quality assurance approach (Chapter 16). If you correct the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs.

3.5 THE CLEANROOM STRATEGY

The philosophy behind cleanroom software engineering is to develop code increments that are right the first time and verify their correctness before testing, rather than relying on costly defect removal processes. Cleanroom software engineering involves the integrated use of software engineering modeling, program verification, and statistical software quality assurance. Under cleanroom software engineering, the analysis and design models are created using a box structure representation (black-box, state box, and clear box). A box encapsulates some system component at a specific level of abstraction. Correctness verification is applied once the box structure design is complete. Once correctness has been verified for each box structure, statistical usage testing commences. This involves defining a set of usage scenarios and determining the probability of use for each scenario. Random data is generated which conform to the usage probabilities. The resulting error records are analysed, and the reliability of the software is determined for the software component.

Distinguishing Characteristics of Cleanroom Techniques

- Makes extensive use of statistical quality control
- Verifies design specification using mathematically-based proof of correctness
- Relies heavily on statistical use testing to uncover high impact errors

Cleanroom Strategy

- Increment planning. The project plan is built around the incremental strategy.
- Requirements gathering. customer requirements are refined for each increment.
- Box structure specification. Box structures isolate and separate the definition of behaviour, data, and procedures at each level of refinement.
- Formal design. Specifications (black-boxes) are iteratively refined to become architectural designs (state-boxes) and component-level designs (clear boxes).
- Correctness verification. Correctness questions are asked and answered and followed by formal mathematical verification when required.
- Code generation, inspection, verification. Box structures are translated into program language; inspections are used to ensure conformance of code and boxes, as well as syntactic correctness of code; followed by correctness verification of the code.
- Statistical test planning. A suite of test cases is created to match the probability distribution of the projected product usage pattern.

- Statistical use testing. A statistical sample of all possible test cases is used rather than exhaustive testing.
- Certification. Once verification, inspection, and usage testing are complete and all defects removed, the increment is certified as ready for integration.

3.6 CHECK YOUR PROGRESS

1. What is the process each manager follows during the life of a project is known as
   A. Project Management
   B. Project Management Life Cycle
   C. Manager life cycle
   D. All of the mentioned
2. What is the abbreviation of PM-CMM?
   A. product management capability maturity model
   B. process management capability maturity model
   C. people management capability maturity model
   D. project management capability maturity model
3. Software Configuration Management can be administered in several ways. These include
   a) A single software configuration management team for the whole organization
   b) A separate configuration management team for each project
   c) Software Configuration Management distributed among the project members
   d) All of the mentioned
4. The main aim of Software Configuration Management (SCM) is _____
   a. Identify change
   b. Control change
   c. To ensure that the change is being properly implemented
   d. All of these
   e. None of these
5. Identify the term which is not related to testing?
   a. Failure
   b. Error
   c. Test Case
   d. Test Bot

6. Which of the following is not a valid phase of SDLC (Software Development Life Cycle)?
    a. Testing Phase
    b. Requirement Phase
    c. Deployment phase
    d. Testing closure

Answers to check your progress:

1. B
2. C
3. A
4. D
5. D
6. D

3.7 SUMMARY

The Management Spectrum focuses on the four P's: people, product, process, and project. It emphasizes on the importance of these 4 parameters in building a successful and robust team and process in software project management.

The software configuration management process identifies the functional and physical attributes of software at critical points in time, and implements procedures to control changes to an identified attribute with the objective of maintaining software integrity and traceability throughout the software life cycle.

Software Testing Strategies include set of guidelines that explains test design and determines how testing needs to be done. Also it includes Components of Test plan Test plan id, features to be tested, test techniques, testing tasks, features pass or fail criteria, test deliverables, responsibilities, and schedule, etc.

3.8 KEYWORDS

- **Project scope:** It is the common understanding among stakeholders about what goes into a project and the factors that define its success.
- **Problem decomposition:** It is an activity that sits at the core of software requirements analysis.
- **Stakeholders**: They are those with an interest in your project's outcome.

- **Software Configuration Management:** This is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.

3.9 QUESTIONS FOR SELF STUDY

1. Who are the different categories people who play a major role in Management spectrum? Explain each one's role briefly.
2. Why is it important to define the scope of the product?
3. What are the steps to be followed while defining the process?
4. Explain five-part common sense approach to software projects
5. Describe explain SCM Scenario along with elements.
6. What are the four primary objectives of SCM?
7. Briefly explain the different phases of Configuration Management.
8. Define each of the topic given below:
    a. Big Picture of Software Testing Strategies
    b. Unit Testing and Integration Testing in OO context
    c. Test Strategeis for WebApps
    d. Distinguish between Recovery testing, Security Testing, Performance Testing, Deployment Testing
    e. The cleanroom strategy

3.10 REFERENCES

1. https://ocw.ui.ac.id/pluginfile.php/13396/mod_resource/content/2/Schwalbe%208th%20-%20Chapter%2002.pdf
2. https://www.tpptechnology.com/
3. https://mymanagementguide.com/

UNIT-4: SOFTWARE QUALITY MANAGEMENT

STRUCTURE

4.0 OBJECTIVES

After studying this unit, you will be able to:

- Describe quality management processes, SQA concepts and principles
- Distinguish between various activities of quality planning, quality management and quality control
- Understand the importance and standards of quality management process and their impact on final products.

4.1 INTRODUCTION

In this unit, we are going to discuss about software quality management. A wide variety of software quality dimensions and factors have been proposed over the years. All try to define a set of characteristics that, if achieved, will lead to high software quality. McCall's and the ISO 9126 quality factors establish characteristics such as reliability, usability, maintainability, functionality, and portability as indicators that quality exists.

4.2 QUALITY CONCEPTS

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc., for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Software Quality Assurance (SQA) is a planned and systematic pattern of activities that are necessary to provide a high degree of confidence regarding quality of a product. It actually provides or gives a quality assessment of quality control activities and helps in determining validity of data or procedures for determining quality. It generally monitors software processes and methods that are used in a project to ensure or assure and maintain quality of software.

4.2.1 SOFTWARE QUALITY

The relevancy of Software quality in modern times is increasing like anything. Nowadays software development companies are more focussed on deploying new codes into production even on an hourly basis without any proper software testing. That's creating a chaotic environment in the market.

People often fail to understand that speed has minimal value if there is no quality. let's learn how to ensure software quality in every build

What is software quality? The quality of software can be defined as the ability of the software to function as per user requirement. When it comes to software products it must satisfy all the functionalities written down in the SRS document.

Key aspects that conclude software quality include,

- **Good design** – It's always important to have a good and aesthetic design to please users
- **Reliability** – Be it any software it should be able to perform the functionality impeccably without issues
- **Durability-** Durability is a confusing term, In this context, durability means the ability of the software to work without any issue for a long period of time.
- **Consistency** – Software should be able to perform consistently over platform and devices
- **Maintainability** – Bugs associated with any software should be able to capture and fix quickly and news tasks and enhancement must be added without any trouble
- **Value for money –** customer and companies who make this app should feel that the money spent on this app has not done to waste.

ISO 9126 QUALITY FACTORS

The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

- Functionality. The degree to which the software satisfies stated needs as indicated by the following sub attributes: suitability, accuracy, interoperability, compliance, and security.
- Reliability. The amount of time that the software is available for use as indicated by the following sub attributes: maturity, fault tolerance, recoverability
- Usability. The degree to which the software is easy to use as indicated by the following sub attributes: understandability, learnability, operability.
- Efficiency. The degree to which the software makes optimal use of system resources as indicated by the following sub attributes: time behavior, resource behavior.
- Maintainability. The ease with which repair may be made to the software as indicated by the following sub attributes: analysability, changeability, stability, testability.
- Portability. The ease with which the software can be transposed from one environment to another as indicated by the following sub attributes: adaptability, installability, conformance, replaceability

Like other software quality factors discussed in the preceding subsections, the ISO 9126 factors do not necessarily lend themselves to direct measurement. However, they do provide a worthwhile basis for indirect measures and an excellent checklist for assessing the quality of a system.

TARGETED QUALITY FACTORS

The quality dimensions and factors focus on the software as a whole and can be used as a generic indication of the quality of an application. A software team can develop a set of quality characteristics and associated questions that would probe the degree to which each factor has been satisfied.

For example, McCall identifies usability as an important quality factor. If you were asked to review a user interface and assess its usability, how would you proceed? You might start with the sub attributes suggested by McCall—understandability, learnability, and operability—but what do these mean in a pragmatic sense? To conduct your assessment, you'll need to address specific, measurable (or at least, recognizable) attributes of the interface as given below:

Intuitiveness. The degree to which the interface follows expected usage patterns so that even a novice can use it without significant training.

- Is the interface layout conducive to easy understanding?
- Are interface operations easy to locate and initiate?
- Does the interface use a recognizable metaphor?
- Is input specified to economize key strokes or mouse clicks?
- Does the interface follow the three golden rules?
- Do aesthetics aid in understanding and usage?

Efficiency. The degree to which operations and information can be located or initiated.

- Does the interface layout and style allow a user to locate operations and information efficiently?
- Can a sequence of operations (or data input) be performed with an economy of motion?
- Are output data or content presented so that it is understood immediately?
- Have hierarchical operations been organized in a way that minimizes the depth to which a user must navigate to get something done?

***Robustness*.** The degree to which the software handles bad input data or inappropriate user interaction.

- Will the software recognize the error if data at or just outside prescribed boundaries is input? More importantly, will the software continue to operate without failure or degradation?
- Will the interface recognize common cognitive or manipulative mistakes and explicitly guide the user back on the right track?
- Does the interface provide useful diagnosis and guidance when an error condition (associated with software functionality) is uncovered?

***Richness*.** The degree to which the interface provides a rich feature set.

- Can the interface be customized to the specific needs of a user?
- Does the interface provide a macro capability that enables a user to identify a sequence of common operations with a single action or command?

4.2.2 ACHIEVING SOFTWARE QUALITY

Software quality doesn't just appear. It is the result of good project management and solid software engineering practice. Management and practice are applied within the context of four broad activities that help a software team achieve high software quality: software engineering methods, project management techniques, quality control actions, and software quality assurance.

SOFTWARE ENGINEERING METHODS

If you expect to build high-quality software, you must understand the problem to be solved. You must also be capable of creating a design that conforms to the problem while at the same time exhibiting characteristics that lead to software that exhibits the quality dimensions and factors.

PROJECT MANAGEMENT TECHNIQUES

The impact of poor management decisions on software quality is as follows: if (1) a project manager uses estimation to verify that delivery dates are achievable, (2) schedule dependencies are understood and the team resists the temptation to use short cuts, (3) risk planning is conducted so problems do not breed chaos, software quality will be affected in a

positive way. In addition, the project plan should include explicit techniques for quality and change management.

QUALITY CONTROL

Quality control encompasses a set of software engineering actions that help to ensure that each work product meets its quality goals. Models are reviewed to ensure that they are complete and consistent. Code may be inspected in order to uncover and correct errors before testing commences. A series of testing steps is applied to uncover errors in processing logic, data manipulation, and interface communication. A combination of measurement and feedback allows a software team to tune the process when any of these work products fail to meet quality goals.

QUALITY ASSURANCE

Quality assurance establishes the infrastructure that supports solid software engineering methods, rational project management, and quality control actions—all pivotal if you intend to build high-quality software. In addition, quality assurance consists of a set of auditing and reporting functions that assess the effectiveness and completeness of quality control actions. The goal of quality assurance is to provide management and technical staff with the data necessary to be informed about product quality, thereby gaining insight and confidence that actions to achieve product quality are working. Of course, if the data provided through quality assurance identifies problems, it is management's responsibility to address the problems and apply the necessary resources to resolve quality issues.

4.3 REVIEW TECHNIQUES

Software reviews are a "filter" for the software process. That is, reviews are applied at various points during software engineering and serve to uncover errors and defects that can then be removed. Software reviews "purify" software engineering work products, including requirements and design models, code, and testing data.

4.3.1 COST IMPACT OF SOFTWARE DEFECTS

Within the context of the software process, the terms defect and fault are synonymous. Both imply a quality problem that is discovered after the software has been released to end users (or to another framework activity in the software process). The term error depicts a quality

problem that is discovered by software engineers (or others) before the software is released to the end user (or to another framework activity in the software process).

The primary objective of technical reviews is to find errors during the process so that they do not become defects after release of the software. The obvious benefit of technical reviews is the early discovery of errors so that they do not propagate to the next step in the software process. A number of industry studies indicate that design activities introduce between 50 and 65 percent of all errors (and ultimately, all defects) during the software process. However, review techniques have been shown to be up to 75 percent effective in uncovering design flaws. By detecting and removing a large percentage of these errors, the review process substantially reduces the cost of subsequent activities in the software process.

4.3.2 REVIEW METRICS AND THEIR USE

Technical reviews are one of many actions that are required as part of good software engineering practice. Each action requires dedicated human effort, Since available project effort is finite, it is important for a software engineering organization to understand the effectiveness of each action by defining a set of metrics that can be used to assess their efficacy.

Although many metrics can be defined for technical reviews, a relatively small subset can provide useful insight. The following review metrics can be collected for each review that is conducted:

Preparation effort, **Ep**—the effort (in person-hours) required to review a work product prior to the actual review meeting

Assessment effort, ***Ea***—the effort (in person-hours) that is expended during the actual review

Rework effort, ***Er***—the effort (in person-hours) that is dedicated to the correction of those errors uncovered during the review

Work product size, ***WPS***—a measure of the size of the work product that has been reviewed (e.g., the number of UML models, or the number of document pages, or the number of lines of code)

Minor errors found, $\boldsymbol{Err_{minor}}$—the number of errors found that can be categorized as minor (requiring less than some prespecified effort to correct)

Major errors found, $\boldsymbol{Err_{major}}$—the number of errors found that can be categorized as major (requiring more than some prespecified effort to correct)

These metrics can be further refined by associating the type of work product that was reviewed for the metrics collected.

4.3.3 ANALYZING METRICS

Before analysis can begin, a few simple computations must occur. The total review effort and the total number of errors discovered are defined as:

$E_{review} = E_p + E_a + E_r$

$Err_{tot} = Err_{minor} + Err_{major}$

Error density represents the errors found per unit of work product reviewed.

Error density = Err_{tot} / WPS

For example, if a requirements model is reviewed to uncover errors, inconsistencies, and omissions, it would be possible to compute the error density in a number of different ways. The requirements model contains 18 UML diagrams as part of 32 overall pages of descriptive materials. The review uncovers 18 minor errors and 4 major errors. Therefore, Errtot 22. Error density is 1.2 errors per UML diagram or 0.68 errors per requirements model page.

If reviews are conducted for a number of different types of work products (e.g., requirements model, design model, code, test cases), the percentage of errors uncovered for each review can be computed against the total number of errors found for all reviews. In addition, the error density for each work product can be computed.

Once data are collected for many reviews conducted across many projects, average values for error density enable you to estimate the number of errors to be found in a new (as yet unreviewed document). For example, if the average error density for a requirements model is 0.6 errors per page, and a new requirement model is 32 pages long, a rough estimate suggests that your software team will find about 19 or 20 errors during the review of the document. If you find only 6 errors, you've done an extremely good job in developing the requirements model or your review approach was not thorough enough.

4.3.4 INFORMAL REVIEWS

Informal reviews include a simple desk check of a software engineering work product with a colleague, a casual meeting (involving more than two people) for the purpose of reviewing a work product, or the review-oriented aspects of pair programming.

A simple desk check or a casual meeting conducted with a colleague is a review. However, because there is no advance planning or preparation, no agenda or meeting structure, and no follow-up on the errors that are uncovered, the effectiveness of such reviews is considerably lower than more formal approaches. But a simple desk check can and does uncover errors that might otherwise propagate further into the software process

Pair programming can be characterized as a continuous desk check. Rather than scheduling a review at some point in time, pair programming encourages continuous review as a work product (design or code) is created. The benefit is immediate discovery of errors and better work product quality as a consequence.

4.3.5 FORMAL TECHNICAL REVIEWS

A formal technical review (FTR) is a software quality control activity performed by software engineers (and others).

The objectives of an FTR are:

- To uncover errors in function, logic, or implementation for any representation of the software;
- To verify that the software under review meets its requirements; (
- To ensure that the software has been represented according to predefined standards;
- To achieve software that is developed in a uniform manner; and
- To make projects more manageable. In addition, the FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation.

The FTR also serves to promote backup and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen. The FTR is actually a class of reviews that includes *walkthroughs and inspections*. Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended. In the sections that follow, guidelines similar to those for a walkthrough are presented as a representative formal technical review.

THE REVIEW MEETING

Regardless of the FTR format that is chosen, every review meeting should abide by the following constraints:

- Between three and five people (typically) should be involved in the review.
- Advance preparation should occur but should require no more than two hours of work for each person.
- The duration of the review meeting should be less than two hours.

Given these constraints, it should be obvious that an FTR focuses on a specific (and small) part of the overall software. For example, rather than attempting to review an entire design, walkthroughs are conducted for each component or small group of components. By narrowing the focus, the FTR has a higher likelihood of uncovering errors.

The review meeting is attended by the review leader, all reviewers, and the producer. One of the reviewers takes on the role of a recorder, that is, the individual who records (in writing) all important issues raised during the review. The FTR begins with an introduction of the agenda and a brief introduction by the producer. The producer then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation. When valid problems or errors are discovered, the recorder notes each.

At the end of the review, all attendees of the FTR must decide whether to:

(1) accept the product without further modification,
(2) reject the product due to severe errors (once corrected, another review must be performed), or
(3) accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required).

After the decision is made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team's findings.

REVIEW REPORTING AND RECORD KEEPING

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting, and a review issues list is produced. In addition, a formal technical review summary report is completed.

A review summary report answers three questions:

1. What was reviewed?
2. Who reviewed it?
3. What were the findings and conclusions?

The review summary report is a single page form (with possible attachments). It becomes part of the project historical record and may be distributed to the project leader and other interested parties.

The review issues list serves two purposes:

(1) To identify problem areas within the product and

(2) To serve as an action item checklist that guides the producer as corrections are made. An issues list is normally attached to the summary report that should be ensured with follow up procedure.

REVIEW GUIDELINES

Guidelines for conducting formal technical reviews must be established in advance, distributed to all reviewers, agreed upon, and then followed. A review that is uncontrolled can often be worse than no review at all.

The following represents a minimum set of guidelines for formal technical reviews:

1. *Review the product, not the producer.* An FTR involves people and egos. Conducted properly, the FTR should leave all participants with a warm feeling of accomplishment. Conducted improperly, the FTR can take on the aura of an inquisition. Errors should be pointed out gently; the tone of the meeting should be loose and constructive; the intent should not be to embarrass or belittle. The review leader should conduct the review meeting to ensure that the proper tone and attitude are maintained and should immediately halt a review that has gotten out of control.
2. *Set an agenda and maintain it.* One of the key maladies of meetings of all types is drift. An FTR must be kept on track and on schedule. The review leader is chartered with the responsibility for maintaining the meeting schedule and should not be afraid to nudge people when drift sets in.

3. *Limit debate and rebuttal.* When an issue is raised by a reviewer, there may not be universal agreement on its impact. Rather than spending time debating the question, the issue should be recorded for further discussion off-line.

4. *Enunciate problem areas,* but don't attempt to solve every problem noted. A review is not a problem-solving session. The solution of a problem can often be accomplished by the producer alone or with the help of only one other individual. Problem solving should be postponed until after the review meeting.

5. *Take written notes.* It is sometimes a good idea for the recorder to make notes on a wall board, so that wording and priorities can be assessed by other reviewers as information is recorded. Alternatively, notes may be entered directly into a notebook computer.

6. *Limit the number of participants and insist upon advance preparation*. Two heads are better than one, but 14 are not necessarily better than 4. Keep the number of people involved to the necessary minimum. However, all review team members must prepare in advance. Written comments should be solicited by the review leader (providing an indication that the reviewer has reviewed the material).

7. *Develop a checklist for each product that is likely to be reviewed.* A checklist helps the review leader to structure the FTR meeting and helps each reviewer to focus on important issues. Checklists should be developed for analysis, design, code, and even testing work products.

8. *Allocate resources and schedule time for FTRs.* For reviews to be effective, they should be scheduled as tasks during the software process. In addition, time should be scheduled for the inevitable modifications that will occur as the result of an FTR.

9. *Conduct meaningful training for all reviewers.* To be effective all review participants should receive some formal training. The training should stress both process-related issues and the human psychological side of reviews. Freedman and Weinberg [Fre90] estimate a one-month learning curve for every 20 people who are to participate effectively in reviews.

10. *Review your early reviews.* Debriefing can be beneficial in uncovering problems with the review process itself. The very first product to be reviewed should be the review guidelines themselves.

Because many variables (e.g., number of participants, type of work products, timing and length, specific review approach) have an impact on a successful review, a software organization should experiment to determine what approach works best in a local context.

4.4 SOFTWARE QUALITY ASSURANCE - SQA

Software quality assurance (SQA) encompasses

(1) An SQA process,

(2) Specific quality assurance and quality control tasks (including technical reviews and a multitiered testing strategy),

(3) Effective software engineering practice (methods and tools),

(4) Control of all software work products and the changes made to them,

(5) A procedure to ensure compliance with software development standards (when applicable), and

(6) Measurement and reporting mechanisms.

4.4.1 ELEMENTS OF SQA

Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality.

Standards. The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

Reviews and audits. Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit of the review process might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.

Testing. Software testing is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

Error/defect collection and analysis. The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.

Change management. Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that adequate change management practices have been instituted.

Education. Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.

Vendor management. Three categories of software are acquired from external software vendors—shrink-wrapped packages (e.g., Microsoft Office), a tailored shell that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and contracted software that is custom designed and constructed from specifications provided by the customer organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible), and incorporating quality mandates as part of any contract with an external vendor.

Security management. With the increase in cyber crime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for WebApps, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security

Safety. Because software is almost always a pivotal component of human rated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.

Risk management. Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

In addition to each of these concerns and activities, SQA works to ensure that software support activities (e.g., maintenance, help lines, documentation, and manuals) are conducted or produced with quality as a dominant concern.

4.5 CHECK YOUR PROGRESS

1. What is software quality?
2. What are the methods to achieve software quality?
3. What is SQA?
4. What are the different review techniques followed to maintain software quality?

Answers to check your progress:

1. The quality of software can be defined as the ability of the software to function as per user requirement. When it comes to software products it must satisfy all the functionalities written down in the SRS document.
2. Software Engineering Methods, Project Management Techniques, Quality Control, Quality Assurance.
3. Software quality assurance (SQA) encompasses
   (1) an SQA process,
   (2) specific quality assurance and quality control tasks (including technical reviews and a multitiered testing strategy),
   (3) effective software engineering practice (methods and tools),
   (4) control of all software work products and the changes made to them,
   (5) a procedure to ensure compliance with software development standards (when applicable), and
   (6) Measurement and reporting mechanisms.
4. The different review techniques are:
   (1) Cost Impact of Software Defects
   (2) Review Metrics and their Use
   (3) Analyzing Metrics
   (4) Informal Reviews
   (5) Formal Technical Reviews
   (6) The Review Meeting
   (7) Review Reporting and Record Keeping

4.6 SUMMARY

A wide variety of software quality dimensions and factors have been proposed over the years. All try to define a set of characteristics that, if achieved, will lead to high software quality. McCall's and the ISO 9126 quality factors establish characteristics such as reliability, usability, maintainability, functionality, and portability as indicators that quality exists.

The intent of every technical review is to find errors and uncover issues that would have a negative impact on the software to be deployed. The sooner an error is uncovered and corrected, the less likely that error will propagate to other software engineering work products and amplify itself, resulting in significantly more effort to correct it.

4.7 KEYWORDS

- **Testing:** It is a quality control function that has one primary goal to find errors.
- **Technical reviews:** This is a quality control activity performed by software engineers for software engineers.
- **Quality control:** It encompasses a set of software engineering actions that help to ensure that each work product meets its quality goals.
- **Durability:** It means the ability of the software to work without any issue for a long period of time.

4.8 QUESTIONS FOR SELF STUDY

1. What are the key aspects that conclude software quality?
2. What are the targeted software quality factors?
3. Discuss any two major techniques to achieve software quality.
4. Explain any 4 review techniques.
5. Write a note on SQA elements.

4.9 REFERENCES

1. https://ocw.ui.ac.id/pluginfile.php/13396/mod_resource/content/2/Schwalbe%208th%20-%20Chapter%2002.pdf
2. https://www.tpptechnology.com/
3. https://mymanagementguide.com/

Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA IT Specialization

IV Semester

MBSC-4.1G Software Project Management

Block 2

PREFACE

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products, . . . the list is almost endless. Software is virtually inescapable in a modern world. And as we move into the twenty-first century, it will become the driver for new advances in everything from elementary education to genetic engineering.

When a computer software succeeds—when it meets the needs of the people who use it, when it performs flawlessly over a long period of time, when it is easy to modify and even easier to use—it can and does change things for the better. But when software fails—when its users are dissatisfied, when it is error prone, when it is difficult to change and even harder to use—bad things can and do happen. We all want to build software that makes things better, avoiding the bad things that lurk in the shadow of failed efforts. To succeed, we need discipline when software is designed and built. We need an engineering approach.

The whole material is organized into four modules each with four units. Each unit lists the objectives of the study along with the relevant questions, illustrations and suggested reading to better understand the concepts.

Wish you happy reading!!!

KARNATAKA STATE OPEN UNIVERSITY

MUKTHAGANGOTRI, MYSURU-06

Dept. of Studies and Research in Management

MBA IT

IV SEMESTER

MBSC-4.1G： SOFTWARE PROJECT MANAGEMENT (4 Credits)

BLOCK 2: Project Scheduling and Management

BLOCK 2 INTRODUCTION

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

Software project management involves many activities, including project planning, software product scope, cost estimation in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

Project Planning

Scope Management

Project Estimation

This block consists of four units and is organized as follows:

UNIT 5 speaks about the scheduling techniques to be allowed and applied in project management to reach the expected deliverable time as instructed by the client. There are several methodologies that can be applied while allotting the different time frames to different sources making sure of no waste of time and productive utilization.

UNIT 6 focuses on Risk Management to make sure there is no any gap in realizing the requirements and reaching the deadlines failing which might result in rework of project or rejection.

UNIT 7 says about the importance of maintenance and reengineering. Every software needs to undergo Updation on a regular basis even after the delivery.

UNIT 8 is about creation and maintenance of relationship with external resources to complete the project. It focuses on required functionality to manage or complete project procurement for any size or type of the project.

UNIT-5: PROJECT SCHEDULING

STRUCTURE

5.0 OBJECTIVES

After studying this unit, you will be able to:

- Describe a work break down structure of project to identify the schedules and planning.
- Identify task relationships and plan accordingly.
- Estimate work packages and calculate the initial schedule.
- Assign and level resources after completion of this unit.

5.1 INTRODUCTION

In this unit, we are going to discuss about project scheduling. Project schedule simply means a mechanism that is used to communicate and know about that tasks are needed and has to be done or performed and which organizational resources will be given or allocated to these tasks and in what time duration or time frame work is needed to be performed.

5.2 PROJECT SCHEDULING - BASIC CONCEPTS AND PRINCIPLES

A comprehensive process of designing a project schedule that outlines the project phases, tasks under each stage, and dependencies is known as project scheduling. It also considers skills and the number of resources required for each task, their order of occurrence, milestones, interdependencies, and timeline.

Furthermore, it involves analysing the resource availability and implementing the scheduling technique to ascertain timely delivery while maintaining the resource health index. Many project managers successfully generate the right schedule, yet most of them find it challenging to manage the resources intelligently.

It can cause delays and discrepancies in the deliverables as their talent pool is responsible for executing these tasks. Thus, they must master each aspect of project scheduling.

5.2.1 BENEFITS OF PROJECT SCHEDULING

1. It brings together all the project-related information in one place that opens doors for seamless communication between the project manager and stakeholders.
2. Project scheduling also enables task prioritization. The initial steps of project scheduling comprise forming a work breakdown structure and dividing the project into simpler tasks. Once the tasks are enlisted, the project manager can implement the

appropriate technique to evaluate the criticality of the tasks and arrange them in order of precedence.

3. In addition, the detailed description of each task and skill demand against them makes it easy for managers to procure the right resources for the right job. Not just that, with real-time information of the project's progress, they can gauge the resource performance and take remedial measures in case of any inconsistencies.
4. The internal team conflicts are minimized when the entire team, stakeholders, and managers are on the same page. Resources are aware of the task dependencies and work diligently to ensure that the overall delivery is not affected.
5. When managers opt for sophisticated scheduling software, they get real-time updates on every project metric, which promotes proactive planning, monitoring, and coherent risk management.

5.2.2 AN OVERVIEW OF DIFFERENT SCHEDULING TECHNIQUES

Project scheduling techniques are beneficial to secure the project timeline and budget without over or underutilizing the workforce. Your resource pool is the success driver of the project, and thus, it is vital to keep their productivity and well-being in check. These scheduling techniques come in handy to ensure that no resource is burned out or sitting idle.

Here's a rundown of some of these techniques:

5.2.2.1 Critical Path Method (CPM)

This technique is purely based on mathematical analysis and lets you calculate the longest and shortest possible project timeline.

Let's understand this better with an example. There are four tasks in the project – A, B, C, and D. Task B and D can only begin after task A completes, whereas task C has no such restriction.

In this case, since the progress of B and D banks on task A, it becomes the critical task. Task A will be time-sensitive as any delay in its completion can delay the entire project's course. On the other hand, given that task C has no dependencies; it can be accomplished within a flexible deadline. Task C, in this case, will have a float time (also referred to as 'slack'). A float-time is where one can prolong a task to a specific limit without impacting the overall project.

This is how a manager can calculate each task's start and finish time, keeping in mind the reliance and coming to a precise conclusion. A CPM technique is applicable to project tasks when all the deliverables and interdependencies are clear.

5.2.2.2 Crashing

Crashing is a tricky duration-compression technique. It involves adding more resources to specific tasks to expedite the project's delivery.

For instance, you have two developers working on your software development project. Your project manager will add one more so that he/she works on the remaining designs. However, this may not always work in your favour as it adds additional responsibilities of collaboration and communication with other resources.

Another way of using the crashing technique is by adding more duration per day and paying overtime to the workforce. It can have two repercussions; cost-escalation which can eat up from your revenue, and two, it can burn out your crew. Thus, crashing can only be applied when it fits your project budget and when you have generic resources as a backup.

5.2.2.3 Simulation

On the contrary to the critical path method, simulation is used when the project attributes like deliverables and interdependencies are unclear. Simulation allows you to gauge multiple scenarios by changing one or two variables.

For example, if one task's timeline is known, you can modify the resource utilization and see how it affects the end date. After trying multiple variations, you can come to the best-fit scenario.

Project managers commonly use the Monte-Carlo simulation model. The advantage is that instead of assigning one constant to the unknown variable, managers can play with multiple values and peruse the results. It facilitates them to come to a best-fit estimation or prediction and form a definite project plan. It also warns you of the potential risks, which give you enough leeway to create a backup plan.

5.2.2.4 Resource optimization

Workforce, their skills, and their effort drive the project's success. Thus, managers must tap into the right potential and leverage their talent to the maximum extent. At the same time, keeping their utilization in check is of utmost importance. If resources are underutilized, it causes lower productivity and unplanned attrition. Overutilization, on the other hand, can cause fatigue and burnout.

This is why, if the timelines of the booked resources stretch thinly, managers deploy an appropriate resource optimization technique. Suppose the project is time-sensitive, and the deadlines cannot be adjusted. In that case, they will book more resources to assist the critical employee or utilize the slack without interrupting the critical path. This is known as resource smoothing. If the project's timeline is adjustable, managers will initiate the project based on the resource's availability to prevent overutilization. This technique is termed resource-levelling.

Implementing these two techniques can help managers define the schedules that align with resources' mental and physical well-being.

5.2.3 STEPS TO FORM AN IDEAL PROJECT SCHEDULE

Developing a project schedule framework is a sequential process consisting of several steps. Based on project requirements or organizational needs, these can be altered. To form a standardized procedure, here are the seven steps you can leverage and create a standard project schedule:

5.2.3.1 Define the project scope

Project scope is the foundation of any project. It distinctly clarifies project goals and objectives, deliverables, features, tasks, budget, and other necessary elements. Project managers can define the scope using a work breakdown structure that allows a systematic division of tasks and timelines.

Defining the scope sets the tone for the entire project schedule as it allows you to understand the minute details before going a step forward.

5.2.3.2 Decide the milestones

Now that the tasks are defined, the next step is to put them in different phases. For instance, you are managing a construction project. The first phase will be forming a plan that constitutes various tasks like designing the blueprint, studying the site, etc. Similarly, the consecutive tasks will be clubbed under relevant phases.

The completion of each phase acts as a checkpoint in the project, known as milestones. Defining the milestones makes it convenient for project managers to track the project's progress and give a sense of accomplishment to the resources.

5.2.3.3 Illustrate task interdependencies

During a project's tenure, many tasks take place simultaneously, while some can only kick-start once the task-in-hand is complete. It is known as interdependence. Taking the same example, only when the design and conception are done, employees can go and assemble the construction material.

It will allow managers to gauge the timeline precisely by giving them enough information on which tasks can be done in parallel and which ones are interdependent. Managers can use the critical path method technique to form the right roadmap of independent and dependent project tasks. This will also allow them to deduce accurate task duration.

5.2.3.4 Assess the resource demand and their availability

Once the roadmap is ready, the next and most crucial step is to understand each task diligently, along with the skillset and number of resources it will require. Managers also need to consider the duration of each task. Analysing the skill demand lets managers find the best-fit resource for it and do justice to the project.

This is not it; project managers have to see the availability of every resource to avoid overbooking or double-booking. If the resource is unavailable, managers will place a formal request to the resource managers to fulfil the demand in due time.

5.2.3.5 Form a resource plan using the right scheduling technique

Once the resource demand is met, managers can finally formulate a project resource plan. It is done before scheduling to ensure that resources are not over or underutilized. For example, you have a critical task that can only be accomplished by highly specialized personnel. He/she is unavailable during the same time. What would you do?

You can implement the correct resource optimization technique based on the timeline constraints of the project. This will ensure that the workforces' schedule is given due importance, and they will be at their productive best when they have only one project to focus on. Even if they are catering to two or more projects, optimizing their schedule will allow them to have a balanced timeline and negate the chances of schedule overruns. Eventually, it will prevent employee burnout and enhance employee satisfaction.

5.2.3.6 Build a contingency plan

Risks are an inevitable part of any project, and they can be in any form. Maybe one of your critical resources takes an unplanned leave, or the project's cost escalates due to procurement

of equipment at the last minute, or so on. In this case, managers have to keep a backup plan in place to keep the project going.

For instance, having a backup resource ready (generic resource) will not cause any delay in the entire project's lifecycle. Here, running the simulation technique, also known as 'what-if analysis, is an ideal choice. It provides outputs to multiple scenarios, and managers can then form a contingency or an action plan to deal with potential bottlenecks in advance.

5.2.3.7 Monitor, review, and update

Last but not least, after forming the schedule, assigning the right resource to the right job, and head-starting the project, it's time to monitor and control the progress. Managers can equip an intuitive project management solution that provides real-time reports on different project metrics like cost, actual vs. forecasted timeline, etc.

Furthermore, they can pair it with a robust resource management tool to get real-time resource performance updates alongside the project. These tools will forewarn the managers in case of discrepancies and proactively take remedial measures. Eventually, it will be easier for them to safeguard the project from going downhill.

Given the market volatility, the project schedule might change due to ad hoc demands. In this case, managers will update the plan in real-time to incorporate the changes and keep everyone informed.

After completing the steps mentioned above, a project schedule is finally ready to give a jumpstart to the project. Out of all these, resource planning is a cornerstone of successful and timely project delivery.

5.3 DEFINING A TASK SET FOR THE SOFTWARE PROJECT

A task set is a collection of software engineering work tasks, milestones, work products, and quality assurance filters that must be accomplished to complete a particular project. The task set must provide enough discipline to achieve high software quality. But, at the same time, it must not burden the project team with unnecessary work.

In order to develop a project schedule, a task set must be distributed on the project time line. The task set will vary depending upon the project type and the degree of rigor with which the software team decides to do its work. Although it is difficult to develop a comprehensive taxonomy of software project types, most software organizations encounter the following projects:

1. Concept development projects that are initiated to explore some new business concept or application of some new technology.
2. New application development projects that are undertaken as a consequence of a specific customer request.
3. Application enhancement projects that occur when existing software undergoes major modifications to function, performance, or interfaces that is observable by the end user.
4. Application maintenance projects that correct, adapt, or extend existing software in ways that may not be immediately obvious to the end user.
5. Reengineering projects that are undertaken with the intent of rebuilding an existing (legacy) system in whole or in part.

Even within a single project type, many factors influence the task set to be chosen.

These include size of the project, number of potential users, mission criticality, application longevity, and stability of requirements, ease of customer/developer communication, and maturity of applicable technology, performance constraints, embedded and non-embedded characteristics, project staff, and reengineering factors.

When taken in combination, these factors provide an indication of the degree of rigor with which the software process should be applied.

5.3.1 A TASK SET EXAMPLE

Concept development projects are initiated when the potential for some new technology must be explored. There is no certainty that the technology will be applicable, but a customer (e.g., marketing) believes that potential benefit exists. Concept development projects are approached by applying the following actions:

1. Concept scoping determines the overall scope of the project.
2. Preliminary concept planning establishes the organization's ability to undertake the work implied by the project scope.
3. Technology risk assessment evaluates the risk associated with the technology to be implemented as part of the project scope.
4. Proof of concept demonstrates the viability of a new technology in the software context.
5. Concept implementation implements the concept representation in a manner that can be reviewed by a customer and is used for "marketing" purposes when a concept must be sold to other customers or management.

6. Customer reaction to the concept solicits feedback on a new technology concept and targets specific customer applications.

A quick scan of these actions should yield few surprises. In fact, the software engineering flow for concept development projects (and for all other types of projects as well) is little more than common sense.

5.3.2 REFINEMENT OF SOFTWARE ENGINEERING ACTIONS

Refinement begins by taking each action and decomposing it into a set of tasks (with related work products and milestones). As an example of task decomposition, consider Action 5.3.1, Concept Scoping. Task refinement can be accomplished using an outline format, but, here, a process design language approach is used to illustrate the flow of the concept scoping action:

Task definition: Action 5.3.1 Concept Scoping

- Identify need, benefits and potential customers;
- Define desired output/control and input events that drive the application;
    - Begin Task - above task
    - TR: Review written description of need7
    - Derive a list of customer visible outputs/inputs
    - TR: Review outputs/inputs with customer and revise as required;
    - endtask
- Define the functionality/behaviour for each major function;
    - Begin Task
    - TR: Review output and input data objects derived in task 1.2.2;
    - Derive a model of functions/behaviours;
    - TR: Review functions/behaviours with customer and revise as required;
    - endtask
- Isolate those elements of the technology to be implemented in software;
- Research availability of existing software;
- Define technical feasibility;
- Make quick estimate of size;
- Create a scope definition;
- endtask definition: Action 5.3.1

The tasks and subtasks noted in the process design language refinement form the basis for a detailed schedule for the concept scoping action.

5.4 DEFINING A TASK NETWORK

Individual tasks and subtasks have interdependencies based on their sequence. In addition, when more than one person is involved in a software engineering project, it is likely that development activities and tasks will be performed in parallel. When this occurs, concurrent tasks must be coordinated so that they will be complete when later tasks require their work product(s).

A task network, also called an activity network, is a graphic representation of the task flow for a project. It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool. In its simplest form (used when creating a macroscopic schedule), the task network depicts major software engineering tasks.

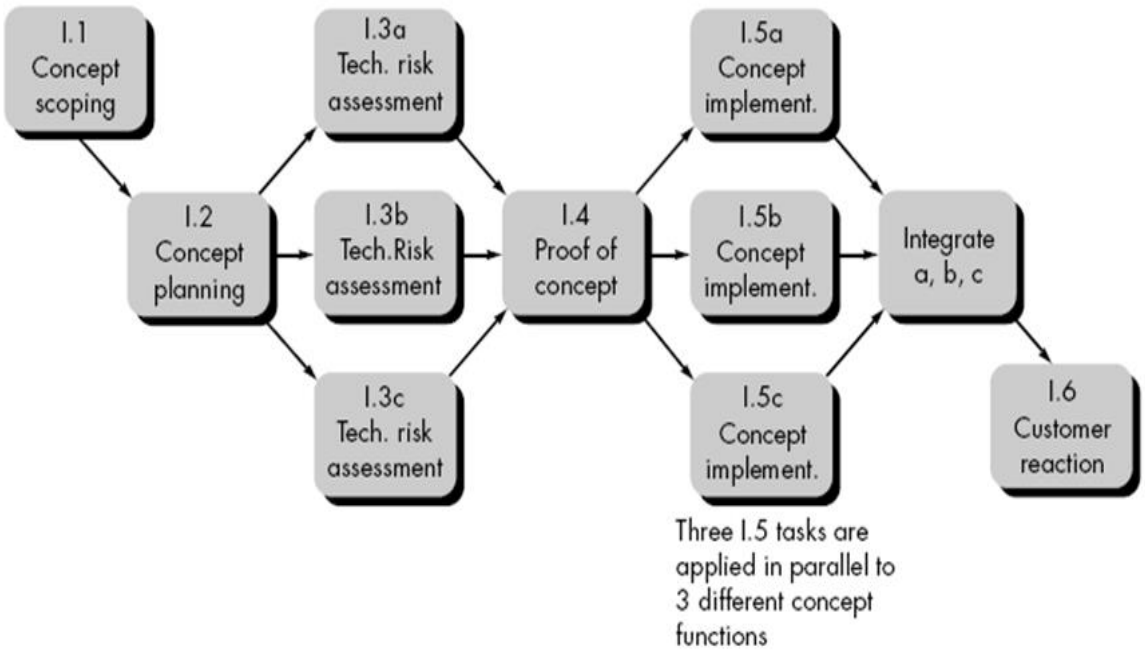


Figure 5.1 – Task Network

The concurrent nature of software engineering activities leads to a number of important scheduling requirements. Because parallel tasks occur asynchronously, the planner must determine inter task dependencies to ensure continuous progress toward completion. In addition, the project manager should be aware of those tasks that lie on the critical path. That is, tasks that must be completed on schedule if the project as a whole is to be completed on schedule.

5.5 SCHEDULING

Program evaluation and review technique (PERT) and the critical path method (CPM) are two project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities: estimates of effort, a decomposition of the product function, the selection of the appropriate process model and task set, and decomposition of the tasks that are selected.

5.5.1 Timeline Charts

A project timeline is a visual list of tasks or activities placed in chronological order, which lets project managers view the entirety of the project plan in one place. A project timeline typically takes the form of a horizontal bar chart, where each task is given a name and a corresponding start and end date.

A project timeline provides an in-depth overview of the entire project from start to finish. You can see when a task starts and when it's due and importantly, whether or not it's dependent on another task.

Project timelines give project managers an opportunity to:

- Organize their tasks
- Show when in the project the tasks start
- View task deadlines
- Link dependent tasks
- Break the project into phases
- Identify team members assigned to a task

5.5.2 Tracking the Schedule

If it has been properly developed, the project schedule becomes a road map that defines the tasks and milestones to be tracked and controlled as the project proceeds.

Tracking can be accomplished in a number of different ways:

- Conducting periodic project status meetings in which each team member reports progress and problems
- Evaluating the results of all reviews conducted throughout the software engineering process
- Determining whether formal project milestones have been accomplished by the scheduled date

- Comparing the actual start date to the planned start date for each project task listed in the resource table
- Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon
- Using earned value analysis to assess progress quantitatively

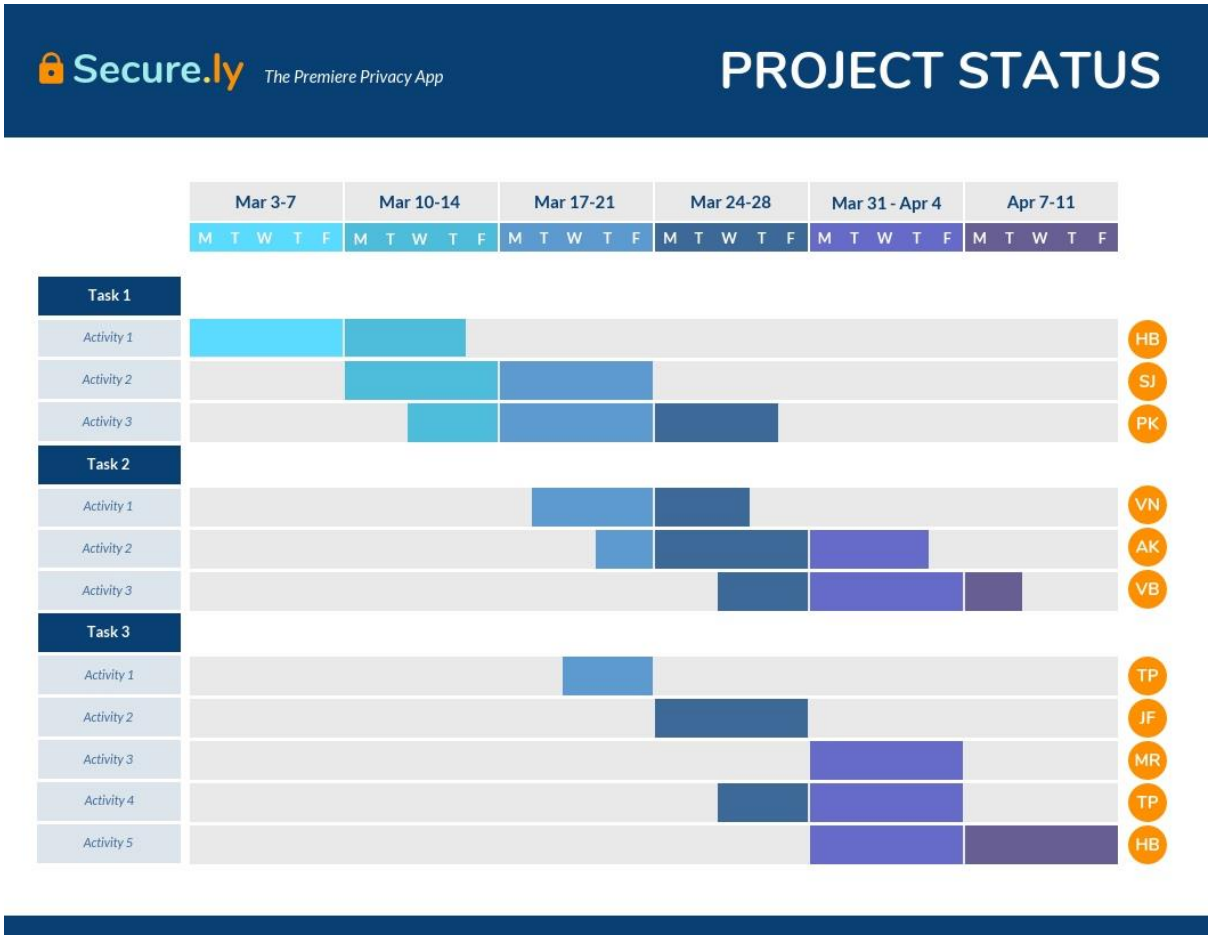


Figure 5.2 – Example Timeline Chart

5.6 EARNED VALUE ANALYSIS

Earned value is a measure of project progress. It enables you to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling. In fact, Fleming and Koppleman argue that earned value analysis "provides accurate and reliable readings of performance from as early as 15 percent into the project." To determine the earned value, the following steps are performed:

The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule. During estimation, the work (in person-hours or person-days) of each software engineering task is planned. Hence, BCWSi is the effort planned for work task

1. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the BCWSi values for all work tasks that should have been completed by that point in time on the project schedule.
2. The BCWS values for all work tasks are summed to derive the budget at completion (BAC). Hence, BAC (BCWSk) for all tasks k.
3. Next, the value for budgeted cost of work performed (BCWP) is computed. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.

Wilkens notes that "the distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed." Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

Schedule performance index, SPI = BCWP/BCWS

Schedule variance, SV BCWP - BCWS

SPI is an indication of the efficiency with which the project is utilizing scheduled resources. An SPI value close to 1.0 indicates efficient execution of the project schedule. SV is simply an absolute indication of variance from the planned schedule.

Percent scheduled for completion= BCWS / BAC provides an indication of the percentage of work that should have been completed by time t.
Percent complete= BCWP / BAC provides a quantitative indication of the percent of completeness of the project at a given point in time t.

It is also possible to compute the actual cost of work performed (ACWP). The value for ACWP is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule. It is then possible to compute

Cost performance index, CPI = BCWP / ACWP

Cost variance, CV = BCWP – ACWP

A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project.

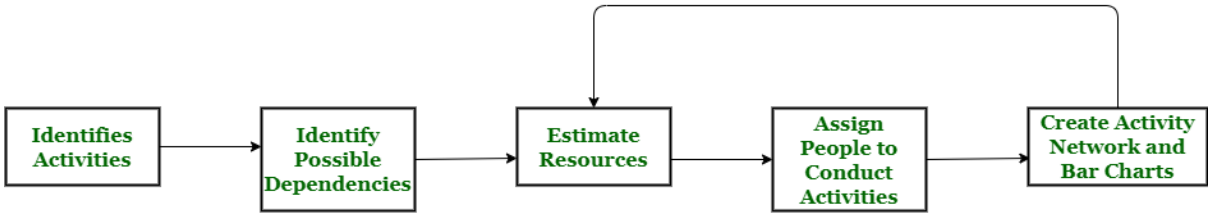
Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables you to take corrective action before a project crisis develops.

5.7 CHECK YOUR PROGRESS

1. What is WBS?
2. Represent Project Scheduling Process with a simple diagram.
3. What are the advantages of Project Scheduling?
4. What is Gantt chart?

Answers to check your progress:

1. A WBS is a hierarchical list of the work activities required to complete a project.
2.



Project Scheduling Process

3. There are several advantages provided by project schedule in our project management:
   - It simply ensures that everyone remains on same page as far as tasks get completed, dependencies, and deadlines.
   - It helps in identifying issues early and concerns such as lack or unavailability of resources.
   - It also helps to identify relationships and to monitor process.
   - It provides effective budget management and risk mitigation.
4. Gantt charts were devised by Henry Gantt (1917). It represents project schedule with respect to time periods. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities.

5.8 SUMMARY

Project schedule simply means a mechanism that is used to communicate and know about that tasks are needed and has to be done or performed and which organizational resources will be given or allocated to these tasks and in what time duration or time frame work is needed to be performed. Effective project scheduling leads to success of project, reduced cost, and increased customer satisfaction. Scheduling in project management means to list out activities, deliverables, and milestones within a project that are delivered. It contains more notes than your average weekly planner notes. The most common and important form of project schedule is Gantt chart.

5.9 KEYWORDS

- **Critical Path Method (CPM):** This technique is purely based on mathematical analysis and lets you calculate the longest and shortest possible project timeline.
- **Crashing:** It is a tricky duration-compression technique. It involves adding more resources to specific tasks to expedite the project's delivery.
- **Project scope:** It is the foundation of any project. It distinctly clarifies project goals and objectives, deliverables, features, tasks, budget, and other necessary elements.
- **Task network:** It is a graphic representation of the task flow for a project.
- **Earned value:** It is a measure of project progress.

5.10 QUESTIONS FOR SELF STUDY

1. Write the purpose and advantages of Project Scheduling.
2. Write a brief overview on different scheduling techniques used in project scheduling with an example.
3. What are the steps to form an ideal project schedule? Explain each one briefly.
4. Define task set and task network for the software project.
5. Explain earned value analysis in brief.

5.11 REFERENCES

1. https://www.sitesbay.com/software-engineering/se-project-management-activities
2. https://www.saviom.com/blog/what-is-project-scheduling-and-why-is-it-important/
3. https://www.projectmanager.com/guides/project-timeline

UNIT-6: RISK MANAGEMENT

STRUCTURE

6.0 OBJECTIVES

After studying this unit, you will be able to:

- Understand the importance of Risk management in Project Management and identify the areas that need Risk management to be performed.
- To identify the Risks across the phases of software development, make mitigation plan for them and manage the same.
- To consider the size of the risk and compare it with the cost of controlling it.
- To identify and manage risks so that there will be less of negative impact on the project in terms of objective attainment.

6.1 INTRODUCTION

In this unit, we are going to discuss about risk management. Risk management is the process of minimizing any potential problems that may negatively impact a project's timetable. 'Risk' is any unexpected event that might affect the people, processes, technology, and resources involved in a project. Unlike 'issues', which are certain to happen, risks are events that could occur, and you may not be able to tell when. Because of this uncertainty, project risk requires preparation in order to manage them efficiently.

6.2 INTRODUCTION TO RISK MANAGEMENT

Project risk management is the process of identifying, analyzing and responding to any risk that arises over the life cycle of a project to help the project remain on track and meet its goal. Risk management isn't reactive only; it should be part of the planning process to figure out the risk that might happen in the project and how to control that risk if it in fact occurs.

A risk is anything that could potentially impact your project's timeline, performance or budget. Risks are potentialities, and in a project management context, if they become realities, they then become classified as "issues" that must be addressed with a risk response plan. So risk management, then, is the process of identifying, categorizing, prioritizing and planning for risks before they become issues.

Risk management can mean different things on different types of projects. On large-scale projects, risk management strategies might include extensive detailed planning for each risk to ensure mitigation strategies are in place if issues arise. For smaller projects, risk management might mean a simple, prioritized list of high, medium and low priority risks.

6.3 REACTIVE VS PROACTIVE RISK STRATEGIES

Reactive risk management tries to reduce the damage of potential threats and speed an organization's recovery from them, but assumes that those threats will happen eventually. Proactive risk management identifies threats and aims to prevent those events from ever happening in the first place.

Each strategy has its own activities, metrics, and behaviours that are useful in risk analysis.

6.3.1 REACTIVE RISK MANAGEMENT

One fundamental point about reactive risk management is that the disaster or threat must occur before management responds. Proactive risk management is all about taking preventative measures before the event to decrease its severity, and that's a good thing to do.

At the same time, however, organizations should develop reactive risk management plans that can be deployed after the event. Otherwise management is making decisions about how to respond as the event happens, which can be a costly and stressful ordeal.

There's an obvious catch with reactive risk management. Although this approach gives you time to understand the risk before acting, you're still always one step behind the unfolding threat. Other projects will lag as you attend to the problem at hand.

6.3.2 HELPING TO WITHSTAND FUTURE RISKS

The reactive approach learns from past or current events and prepares for future events. For example, businesses can purchase "cybersecurity insurance" to cover the costs of a security disruption.

This strategy assumes that a breach will happen at some point. Once that breach does occur, the business might understand more about how to avoid future breaches, and perhaps could even tailor its insurance policies accordingly.

Fundamentally, however, the organization reacts after the threat has occurred and alters its measures to prevent future potential risks.

6.3.3 PROACTIVE RISK MANAGEMENT

As the name suggests, proactive risk management means that you identify risks before they happen and figure out ways to avoid or alleviate the risk. It seeks to reduce the hazard's risk potential or, even better, prevent the threat altogether.

A good example here is vulnerability testing and remediation. Any organization of appreciable size is likely to have vulnerabilities in its software, which attackers could find an exploit. So regular testing (or, even better, continuous testing) can help to repair those vulnerabilities and eliminate that particular threat.

6.3.4 ALLOWS FOR MORE CONTROL OVER RISK MANAGEMENT

Proactive management strategy gives you more control over your risk management generally. You can decide which issues should be top priorities, and what potential damage you're willing to accept.

Proactive management also involves constant monitoring of your systems, risk processes, cybersecurity, competition, business trends, and so forth. By understanding the level of risk prior to an event, you can educate and instruct your employees on how to mitigate them.

A truly proactive approach, however, does imply that each risk is constantly monitored. It also entails regular risk reviews to update the current risk and new risks affecting the company. This approach drives management to be always aware of the direction of those risks.

6.3.5 PREDICTIVE RISK MANAGEMENT

Where does predictive risk management fit in all this? As the name suggests, it's all about predicting future risks, outcomes, and threats. Some predictive components may sound similar to proactive or reactive strategies.

Predictive risk management attempts to:

- Identify probability of risk in a situation, based on one or more variables
- Anticipate potential future risks and their probability
- Anticipate necessary risk controls

6.4 SOFTWARE RISKS

Although there has been considerable debate about the proper definition for software risk, there is general agreement that risk always involves two characteristics: uncertainty—the risk may or may not happen; that is, there are no 100 percent probable risks—and loss—if the risk becomes a reality, unwanted consequences or losses will occur. When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered.

Project risks threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project. Project complexity, size, and the degree of structural uncertainty were also defined as project (and estimation) risk factors.

Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and

maintenance problems. In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also risk factors. Technical risks occur because the problem is harder to solve than you thought it would be.

Business risks threaten the viability of the software to be built and often jeopardize the project or the product. Candidates for the top five business risks are (1) building an excellent product or system that no one really wants (market risk), (2) building a product that no longer fits into the overall business strategy for the company (strategic risk), (3) building a product that the sales force doesn't understand how to sell (sales risk), (4) losing the support of senior management due to a change in focus or a change in people (management risk), and (5) losing budgetary or personnel commitment (budget risks).

It is extremely important to note that simple risk categorization won't always work. Some risks are simply unpredictable in advance.

6.5 RISK IDENTIFICATION

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

There are two distinct types of risks called *generic risks* and *product-specific risks*. Generic risks are a potential threat to every software project. Product-specific risks can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built. To identify product-specific risks, the project plan and the software statement of scope are examined, and an answer to the following question is developed: "What special characteristics of this product may threaten our project plan?"

One method for identifying risks is to create a *risk item checklist*. The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories:

- Product size—risks associated with the overall size of the software to be built or modified.
- Business impact—risks associated with constraints imposed by management or the marketplace

- Stakeholder characteristics—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.
- Process definition—risks associated with the degree to which the software process has been defined and is followed by the development organization.
- Development environment—risks associated with the availability and quality of the tools to be used to build the product.
- Technology to be built—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- Staff size and experience—risks associated with the overall technical and project experience of the software engineers who will do the work.

The risk item checklist can be organized in different ways. Questions relevant to each of the topics can be answered for each software project. The answers to these questions allow you to estimate the impact of risk. A different risk item checklist format simply lists characteristics that are relevant to each generic subcategory. Finally, a set of "risk components and drivers" are listed along with their probability of occurrence.

A number of comprehensive checklists for software project risk are available on the Web. You can use these checklists to gain insight into generic risks for software projects.

6.5.1 ASSESSING OVERALL PROJECT RISK

The following questions have been derived from risk data obtained by surveying experienced software project managers in different parts of the world. The questions are ordered by their relative importance to the success of a project.

1. Have top software and customer managers formally committed to support the project?
2. Are end users enthusiastically committed to the project and the system/ product to be built? 3. Are requirements fully understood by the software engineering team and its customers?
3. Have customers been involved fully in the definition of requirements?
4. Do end users have realistic expectations?
5. Is the project scope stable?
6. Does the software engineering team have the right mix of skills?
7. Are project requirements stable?
8. Does the project team have experience with the technology to be implemented?

9. Is the software project we're working on at serious risk?
10. Is the number of people on the project team adequate to do the job?
11. Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

If any one of these questions is answered negatively, mitigation, monitoring, and management steps should be instituted without fail. The degree to which the project is at risk is directly proportional to the number of negative responses to these questions.

6.5.2 RISK COMPONENTS AND DRIVERS

Risk components are defined in the following manner:

- ***Performance risk***—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- ***Cost risk***—the degree of uncertainty that the project budget will be maintained.
- ***Support risk***—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- ***Schedule risk***—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk driver on the risk component is divided into one of four impact categories—negligible, marginal, critical, or catastrophic. Referring to Figure 6.1, a characterization of the potential consequences of errors (rows labeled 1) or a failure to achieve a desired outcome (rows labeled 2) are described. The impact category is chosen based on the characterization that best fits the description in the table.

6.6 RISK PROJECTION

Risk projection, also called risk estimation, attempts to rate each risk in two ways—

(1) the likelihood or probability that the risk is real and

(2) The consequences of the problems associated with the risk, should it occur.

You work along with other managers and technical staff to perform four risk projection steps:

1. Establish a scale that reflects the perceived likelihood of a risk.
2. Delineate the consequences of the risk.
3. Estimate the impact of the risk on the project and the product.
4. Assess the overall accuracy of the risk projection so that there will be no misunderstandings.

COMPONENTS / CATEGORY		PERFORMANCE SUPPORT	SUPPORT	COST	SCHEDULE
CASTASTROPHIC	1	Failure to meet the requirement would result in mission failure		Failure results inincreased costs and schedule delays with expected values in excess of $500K	
	2	Significant degradation to nonachievement of technical performance	Nonresponsive or unsupportable software	Significant financial shortages, budget overrun likely	Unachievable delivery date
CRITICAL	1	Failure to meet the requirement would degrade system performance to a point where mission success is questionable		Failure results in operational delays and/or increased costs with expected value of $100K to $500K	
	2	Some reduction in technical performance	Minor delays in software modifications	Some shortage of financial resources, possible overruns	Possible slippage in delivery date
MARGINAL	1	Failure to meet the requirement would result in degradationof secondary mission		Costs, impacts, and/or recoverable schedule slips with expected value of $1 to $100k	
	2	Minimal to small reductionin Technical performance	Responsive software support	Sufficient financial resources	Realistic, achievable schedule
NEGLIGIBLE	1	Failure to meet the requirement would create inconvenience or nonperational impact		Error results in minor cost and/or schedule impact with expected value of less than $1K	
	2	No reduction in Technical performance	Easily supportable software	Possible budget underrun	Early achievable delivery date

Figure 6.1 – Impact Assessment

Note: (1) The potential consequence of undetected software errors or faults. (2) The potential consequence if the desired outcome is not achieved.

The intent of these steps is to consider risks in a manner that leads to prioritization. No software team has the resources to address every possible risk with the same degree of rigor. By prioritizing risks, you can allocate resources where they will have the most impact.

6.7 RISK MITIGATION, MONITORING AND MANAGEMENT

All of the risk analysis activities presented to this point have a single goal—to assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues: risk avoidance, risk monitoring, and risk management and contingency planning.

If a software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation. For example, assume that high staff turnover is noted as a project risk r1. Based on past history and management intuition, the likelihood l1 of high turnover is estimated to be 0.70 (70 percent, rather high) and the impact x1 is projected as critical. That is, high turnover will have a critical impact on project cost and schedule.

To mitigate this risk, you would develop a strategy for reducing turnover. Among the possible steps to be taken are:

Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).

- Mitigate those causes that are under your control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner.
- Conduct peer reviews of all work (so that more than one person is “up to speed”).
- Assign a backup staff member for every critical technologist

As the project proceeds, *risk-monitoring* activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the general attitude of team members based on project pressures, the degree to which the team has jelled, interpersonal relationships among team members, potential problems with compensation and benefits, and the availability of jobs within the company and outside it are all monitored.

In addition to monitoring these factors, a project manager should monitor the effectiveness of risk mitigation steps. For example, a risk mitigation step noted here called for the definition of work product standards and mechanisms to be sure that work products are developed in a timely manner. This is one mechanism for ensuring continuity, should a critical individual leave the project. The project manager should monitor work products carefully to ensure that each can stand on its own and that each imparts information that would be necessary if a newcomer were forced to join the software team somewhere in the middle of the project.

Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality. Continuing the example, the project is well under way and a number of people announce that they will be leaving. If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team. In addition, you can temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to "get up to speed." Those individuals who are leaving are asked to stop all work and spend their last weeks in "knowledge transfer mode." This might include video-based knowledge capture, the development of "commentary documents or Wikis," and/or meeting with other team members who will remain on the project.

It is important to note that risk mitigation, monitoring, and management (RMMM) steps incur additional project cost. For example, spending the time to back up every critical technologist costs money. Part of risk management, therefore, is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them. In essence, you perform a classic cost-benefit analysis. If risk aversion steps for high turnover will increase both project cost and duration by an estimated 15 percent, but the predominant cost factor is "backup," management may decide not to implement this step. On the other hand, if the risk aversion steps are projected to increase costs by 5 percent and duration by only 3 percent, management will likely put all into place.

Risk is not limited to the software project itself. Risks can occur after the software has been successfully developed and delivered to the customer. These risks are typically associated with the consequences of software failure in the field.

Software safety and hazard analysis are software quality assurance activities that focus on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software engineering

process, software design features can be specified that will either eliminate or control potential hazards.

6.8 CHECK YOUR PROGRESS

1. Briefly Explain the Roles of a Risk Manager.
2. How to perform Risk Identification?
3. Briefly explain the Process of Risk Management.
4. What is Software safety and hazard analysis?

Answers to check your progress:

1. "Risk Managers identify and evaluate the risks that are likely to be faced by an organization. They come up with ways to control or mitigate risks and liabilities. A risk manager offers practical risk models involving credit, operational, and market risk, guaranteeing operation control. Risk managers also evaluate existing risk handling measures to identify gaps. They develop risk management plans to assess, mitigate, and recover from risks effectively."
2. "Risk identification is a preliminary step in risk management that involves communicating and documenting concerns. Risk identification begins with understanding business objectives. A risk manager must identify undesirable outcomes, unwanted events, emerging opportunities as well as emerging threats.

   The following are steps I would undertake to identify risks: Understanding the core things that should be considered, gathering information, applying tools and techniques of risk identification, and documenting risks."
3. Identifying risk – this is where potential risks that are likely to affect the business are uncovered and described.

   Analyzing risks – here, the risk manager examines each identified risk to understand the magnitude of their impact on organizational goals.

   Risk evaluation – this is where risks are ranked according to the negative effect on an organization.

   Deal with risks – the risk manager develops preventive plans, contingency plans, and risk mitigation strategies. You will respond depending on the risks that have great risk on the business.

   Risk monitoring – at this stage, tracking and reviewing risks is done."

4. Software safety and hazard analysis are software quality assurance activities that focus on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.

6.9 SUMMARY

Risk management is the process of minimizing any potential problems that may negatively impact a project's timetable. 'Risk' is any unexpected event that might affect the people, processes, technology, and resources involved in a project. Unlike 'issues', which are certain to happen, risks are events that could occur, and you may not be able to tell when. Because of this uncertainty, project risk requires preparation in order to manage them efficiently.

Any business, regardless of size or field, can benefit from adopting a systematic plan for dealing with potential threats through a risk management strategy. Instead of viewing risk management strategy as a sequence of discrete tasks, it is more helpful to think of it as an iterative process in which new and existing risks must be continuously detected, analyzed, managed, and monitored. It allows for continuous assessment and response, ensuring that the company's people, property, and resources are always safe.

6.10 KEYWORDS

- **Reactive risk management:** It is a response-based approach to risk.
- **Proactive risk management:** It identifies the root cause of problem to eliminate it from reoccurring.
- **Predictive risk management:** It is the process of performing risk management activities on hypothetical hazards, risk events, and/or consequences.
- **Performance risk:** the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- **Technical risks:** It threatens the quality and timeliness of the software to be produced.

6.11 QUESTIONS FOR SELF STUDY

1. Explain the role and importance of Risk Management in Project Management.

2. Briefly discuss the different steps of Risk Management.

3. Write a sample diagram to show the risk projection with impact assessment.

4. What are the predictable risks that can be identified under the set of generic risks?
5. Write a note on Risk Mitigation, Monitoring and Management.

6.12 REFERENCES

1. https://www.projectmanager.com/blog/risk-management-process-steps
2. https://reciprocity.com/resources/proactive-vs-reactive-risk-management-strategies
3. https://www.wrike.com/project-management-guide/faq/what-is-risk-management-in-project-management/

UNIT-7: MAINTENANCE AND REENGINEERING

STRUCTURE

7.0 OBJECTIVES

After studying this unit, you will be able to:

- Understand the necessity of maintenance criterion of the software to make it more sustainable.
- Get to know a periodical review has to be done on the software to enhance or delete the features of the project to maintain its competency in the market.

- Create a mechanism for optimization, error correction, deletion of discarded features and enhancement of existing features.
- Create an estimate for above mentioned mechanism.

7.1 INTRODUCTION

In this unit, we are going to discuss about maintenance and reengineering. Software maintenance and support are ongoing activities that occur throughout the life cycle of an application. During these activities, defects are corrected, applications are adapted to a changing operational or business environment, enhancements are implemented at the request of stakeholders, and users are supported as they integrate an application into their personal or business workflow.

7.2 SOFTWARE MAINTENANCE

Software maintenance is a part of the Software Development Life Cycle. Its primary goal is to modify and update software application after delivery to correct errors and to improve performance. Software is a model of the real world. When the real world changes, the software require alteration wherever possible.

Software Maintenance is an inclusive activity that includes error corrections, enhancement of capabilities, deletion of obsolete capabilities, and optimization.

Software Maintenance is needed for:-

- Correct errors
- Change in user requirement with time
- Changing hardware/software requirements
- To improve system efficiency
- To optimize the code to run faster
- To modify the components
- To reduce any unwanted side effects.

Thus, the maintenance is required to ensure that the system continues to satisfy user requirements.

7.2.1 TYPES OF SOFTWARE MAINTENANCE

Maintenance can be divided into the following:

- **Corrective maintenance:**

Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.

- **Adaptive maintenance:**

This includes modifications and updates when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

- **Perfective maintenance:**

A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.

- **Preventive maintenance:**

This type of maintenance includes modifications and updates to prevent future problems of the software. It goals to attend problems, which are not significant at this moment but may cause serious issues in future.

7.3 SOFTWARE SUPPORTABILITY - REENGINEERING

Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

Objectives of Re-engineering:

- To describe a cost-effective option for system evolution.
- To describe the activities involved in the software maintenance process.
- To distinguish between software and data re-engineering and to explain the problems of data re-engineering.

Steps involved in Re-engineering:

- Inventory Analysis
- Document Reconstruction

- Reverse Engineering
- Code Reconstruction
- Data Reconstruction
- Forward Engineering

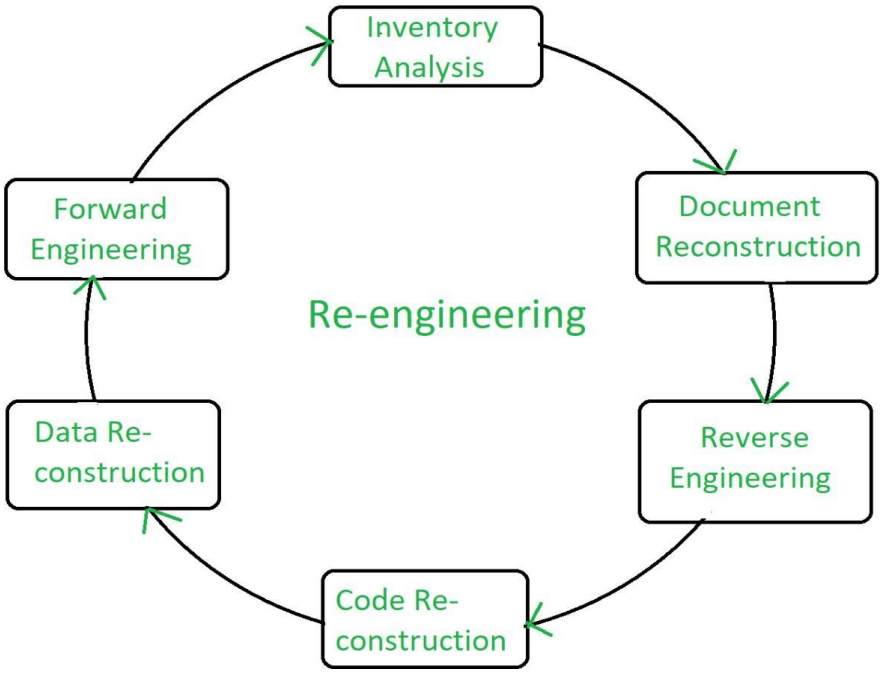


Figure 7.1 – Software Reengineering

Re-engineering Cost Factors:

The quality of the software to be re-engineered

- The tool support available for re-engineering
- The extent of the required data conversion
- The availability of expert staff for re-engineering

Advantages of Re-engineering:

- ***Reduced Risk***: As the software is already existing, the risk is less as compared to new software development. Development problems, staffing problems and specification problems are the lots of problems which may arise in new software development.
- ***Reduced Cost*****:** The cost of re-engineering is less than the costs of developing new software.
- ***Revelation of Business Rules*****:** As a system is re-engineered , business rules that are embedded in the system are rediscovered.

- ***Better use of Existing Staff*:** Existing staff expertise can be maintained and extended accommodate new skills during re-engineering.

Disadvantages of Re-engineering:

Practical limits to the extent of re-engineering.

- Major architectural changes or radical reorganizing of the systems data management has to be done manually.
- Re-engineered system is not likely to be as maintainable as a new system developed using modern software Re-engineering methods.

7.4 BUSINESS PROCESS REENGINEERING

Business process re-engineering is not just a change, but actually it is a dramatic change and dramatic improvements. This is only achieved through overhaul the organization structures, job descriptions, performance management, training and the most importantly, the use of IT i.e. Information Technology.

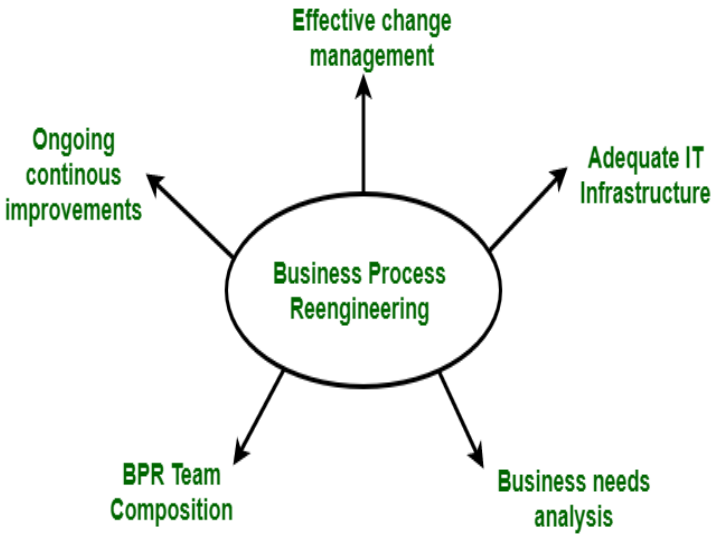


Figure 7.2 – Business Process Reengineering

BPR projects have failed sometimes to meet high expectations. Many unsuccessful BPR attempts are due to the confusion surrounding BPR and how it should be performed. It becomes the process of trial and error.

Phases of BPR:

According to Peter F. Drucker, ” Re-engineering is new, and it has to be done.”

There are 7 different phases for BPR. All the projects for BPR begin with the most critical requirement i.e. communication throughout the organization.

1. Begin organizational change.
2. Build the re-engineering organization.
3. Identify BPR opportunities.
4. Understand the existing process.
5. Reengineer the process
6. Blueprint the new business system.
7. Perform the transformation.

Objectives of BPR:

Following are the objectives of the BPR:

- To dramatically reduce cost.
- To reduce time requirements.
- To improve customer services dramatically.
- To reinvent the basic rules of the business e.g. The airline industry.
- Customer satisfaction.
- Organizational learning.

Challenges faced by BPR process:

All the BPR processes are not as successful as described. The companies that have start the use of BPR projects face many of the following challenges :

- Resistance
- Tradition
- Time requirements
- Cost
- Job losses

Advantages of BPR:

Following are the advantages of BPR:

- BPR offers tight integration among different modules.
- It offers same views for the business i.e. same database, consistent reporting and analysis.

- It offers process orientation facility i.e. streamline processes.
- It offers rich functionality like templates and reference models.
- It is flexible.
- It is scalable.
- It is expandable.

Disadvantages of BPR:

Following are the Disadvantages of BPR:

- It depends on various factors like size and availability of resources. So, it will not fit for every business.
- It is not capable of providing an immediate resolution.

7.5 SOFTWARE REENGINEERING

Software Re-Engineering is the examination and alteration of a system to reconstitute it in a new form. The principles of Re-Engineering when applied to the software development process is called software re-engineering. It affects positively at software cost, quality, service to the customer and speed of delivery. In Software Re-engineering, we are improving the software to make it more efficient and effective.

The need of software Re-engineering: Software re-engineering is an economical process for software development and quality enhancement of the product. This process enables us to identify the useless consumption of deployed resources and the constraints that are restricting the development process so that the development process could be made easier and cost-effective (time, financial, direct advantage, optimize the code, indirect benefits, etc.) and maintainable. The software reengineering is necessary for having-

a) *Boost up productivity*: Software reengineering increase productivity by optimizing the code and database so that processing gets faster.

b) *Processes in continuity:* The functionality of older software product can be still used while the testing or development of software.

c) *Improvement opportunity*: Meanwhile the process of software reengineering, not only software qualities, features and functionality but also your skills are refined, new ideas hit in your mind. This makes the developers mind accustomed to capturing new opportunities so that more and more new features can be developed.

d) *Reduction in risks*: Instead of developing the software product from scratch or from the beginning stage here developers develop the product from its existing stage to enhance some specific features that are brought in concern by stakeholders or its users. Such kind of practice reduces the chances of fault fallibility.

e) *Saves time*: As we stated above here that the product is developed from the existing stage rather than the beginning stage so the time consumes in software engineering is lesser.

f) *Optimization*: This process refines the system features, functionalities and reduces the complexity of the product by consistent optimization as maximum as possible.

Re-Engineering cost factors:

- The quality of the software to be re-engineered.
- The tool support availability for engineering.
- The extent of the data conversion which is required.
- The availability of expert staff for Re-engineering.

7.5.1 SOFTWARE REENGINEERING ACTIVITIES

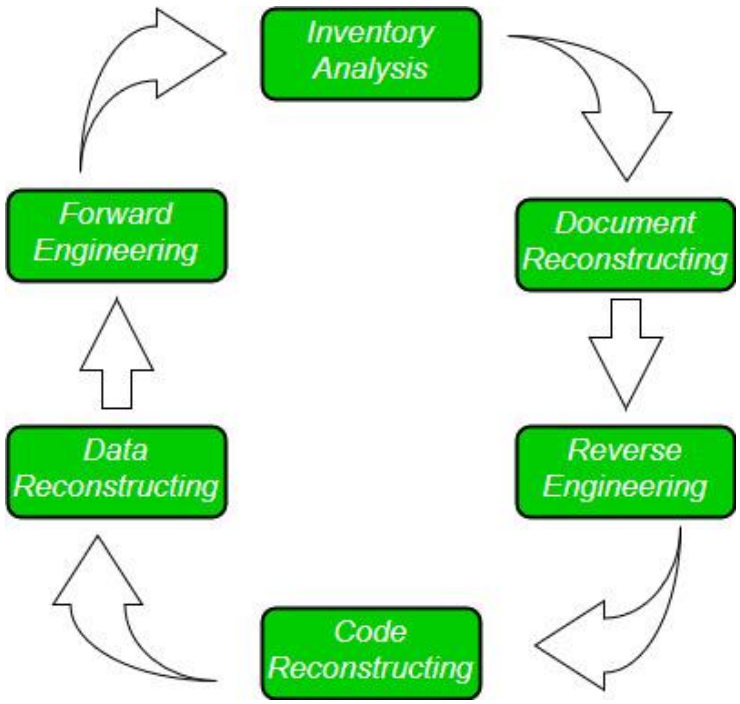


Figure 7.3 – Software Reengineering Activities

1. **Inventory Analysis:**

Every software organisation should have an inventory of all the applications.

- Inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application.
- By sorting this information according to business criticality, longevity, current maintainability and other local important criteria, candidates for re-engineering appear.
- The resource can then be allocated to a candidate application for re-engineering work.

2. **Document reconstructing:**

- Documentation of a system either explains how it operates or how to use it.
- Documentation must be updated.
- It may not be necessary to fully document an application.
- The system is business-critical and must be fully re-documented.

3. **Reverse Engineering:**

Reverse engineering is a process of design recovery. Reverse engineering tools extract data, architectural and procedural design information from an existing program.

4. **Code Reconstructing:**

To accomplish code reconstructing, the source code is analysed using a reconstructing tool. Violations of structured programming construct are noted and code is then reconstructed.

The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.

5. **Data Restructuring:**

Data restructuring begins with a reverse engineering activity. Current data architecture is dissected, and the necessary data models are defined. Data objects and attributes are identified, and existing data structures are reviewed for quality.

6. **Forward Engineering:**

Forward engineering also called as renovation or reclamation not only for recovers design information from existing software but uses this information to alter or reconstitute the existing system in an effort to improve its overall quality.

7.6 REVERSE ENGINEERING

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandbility of a system and to produce the necessary documents for a legacy system.

Reverse Engineering Goals:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.

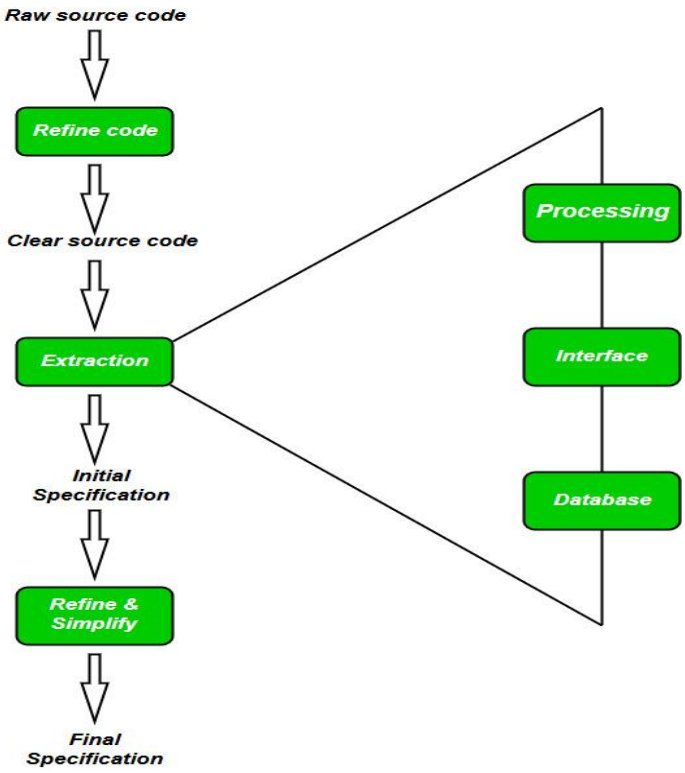


Figure 7.4 – Reverse Engineering Steps

7.6.1 STEPS OF SOFTWARE REVERSE ENGINEERING:

Collection Information:

This step focuses on collecting all possible information (i.e., source design documents etc.) about the software.

- *Examining the information:*

The information collected in step-1 as studied so as to get familiar with the system.

- *Extracting the structure:*

This step concerns with identification of program structure in the form of structure chart where each node corresponds to some routine.

- *Recording the functionality:*

During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.

- *Recording data flow:*

From the information extracted in step-3 and step-4, set of data flow diagrams are derived to show the flow of data among the processes.

- *Recording control flow:*

High level control structure of the software is recorded.

- *Review extracted design:*

Design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.

- *Generate documentation:*

Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. are recorded for future use.

7.6.2 REVERSE ENGINEERING TO UNDERSTAND DATA

Reverse engineering of data occurs at different levels of abstraction and is often the first reengineering task. At the program level, internal program data structures must often be reverse engineered as part of an overall reengineering effort. At the system level, global data structures (e.g., files, databases) are often reengineered to accommodate new database management paradigms (e.g., the move from flat file to relational or object-oriented database systems). Reverse engineering of the current global data structures sets the stage for the introduction of a new system wide database.

Internal data structures. Reverse engineering techniques for internal program data focus on the definition of classes of objects. This is accomplished by examining the program code with the intent of grouping related program variables.

Database structure. Regardless of its logical organization and physical structure, a database allows the definition of data objects and supports some method for establishing relationships

among the objects. Therefore, reengineering one database schema into another requires an understanding of existing objects and their relationships.

7.6.3 REVERSE ENGINEERING TO UNDERSTAND PROCESSING

Reverse engineering to understand processing begins with an attempt to understand and then extract procedural abstractions represented by the source code. To understand procedural abstractions, the code is analyzed at varying levels of abstraction: system, program, component, pattern, and statement.

Things become more complex when the code inside a component is considered. You should look for sections of code that represent generic procedural patterns. In almost every component, a section of code prepares data for processing (within the module), a different section of code does the processing, and another section of code prepares the results of processing for export from the component. Within each of these sections, you can encounter smaller patterns; for example, data validation and bounds checking often occur within the section of code that prepares data for processing.

For large systems, reverse engineering is generally accomplished using a semi-automated approach. Automated tools can be used to help you understand the semantics of existing code. The output of this process is then passed to restructuring and forward engineering tools to complete the reengineering process.

7.6.4 REVERSE ENGINEERING USER INTERFACES

To fully understand an existing user interface, the structure and behaviour of the interface must be specified. Merlo and his colleagues suggest three basic questions that must be answered as reverse engineering of the UI commences:

- What are the basic actions (e.g., keystrokes and mouse clicks) that the interface must process?
- What is a compact description of the behavioral response of the system to these actions?
- What is meant by a “replacement,” or more precisely, what concept of equivalence of interfaces is relevant here?

It is important to note that a replacement GUI may not mirror the old interface exactly (in fact, it may be radically different). It is often worthwhile to develop a new interaction metaphor. For example, an old UI requests that a user provide a scale factor (ranging from 1

to 10) to shrink or magnify a graphical image. A reengineered GUI might use a slide-bar and mouse to accomplish the same function.

7.7 RESTRUCTURING

Software restructuring modifies source code and/or data in an effort to make it amenable to future changes. In general, restructuring does not modify the overall program architecture. It tends to focus on the design details of individual modules and on local data structures defined within modules. If the restructuring effort extends beyond module boundaries and encompasses the software architecture, restructuring becomes forward engineering (Section 7.8).

Restructuring occurs when the basic architecture of an application is solid, even though technical internals need work. It is initiated when major parts of the software are serviceable and only a subset of all modules and data need extensive modification.

7.7.1 CODE RESTRUCTURING

Code restructuring is performed to yield a design that produces the same function but with higher quality than the original program. In general, code restructuring techniques using Boolean algebra and then apply a series of transformation rules that yield restructured logic. The objective is to take "spaghetti-bowl" code and derive a procedural design that conforms to the structured programming philosophy.

Other restructuring techniques have also been proposed for use with reengineering tools. A resource exchange diagram maps each program module and the resources (data types, procedures and variables) that are exchanged between it and other modules. By creating representations of resource flow, the program architecture can be restructured to achieve minimum coupling among modules.

7.7.2 DATA RESTRUCTURING

Before data restructuring can begin, a reverse engineering activity called analysis of source code should be conducted. All programming language statements that contain data definitions, file descriptions, I/O, and interface descriptions are evaluated. The intent is to extract data items and objects, to get information on data flow, and to understand the existing data structures that have been implemented. This activity is sometimes called data analysis.

Once data analysis has been completed, data redesign commences. In its simplest form, a data record standardization step clarifies data definitions to achieve consistency among data item

names or physical record formats within an existing data structure or file format. Another form of redesign, called data name rationalization, ensures that all data naming conventions conform to local standards and that aliases are eliminated as data flow through the system.
When restructuring moves beyond standardization and rationalization, physical modifications to existing data structures are made to make the data design more effective. This may mean a translation from one file format to another, or in some cases, translation from one type of database to another.

7.8 FORWARD ENGINEERING

Over the past few decades, many mainframe applications have been reengineered to accommodate client-server architectures (including WebApps). In essence, centralized computing resources (including software) are distributed among many client platforms. Although a variety of different distributed environments can be designed, the typical mainframe application that is reengineered into a client-server architecture has the following features:

- Application functionality migrates to each client computer.
- New GUI interfaces are implemented at the client sites.
- Database functions are allocated to the server.
- Specialized functionality (e.g., compute-intensive analysis) may remain at the server site.
- New communications, security, archiving, and control requirements must be established at both the client and server sites.

It is important to note that the migration from mainframe to client-server computing requires both business and software reengineering. In addition, an “enterprise network infrastructure” should be established.

Reengineering for client-server applications begins with a thorough analysis of the business environment that encompasses the existing mainframe. Three layers of abstraction can be identified. The database sits at the foundation of a client-server architecture and manages transactions and queries from server applications. Yet these transactions and queries must be controlled within the context of a set of business rules (defined by an existing or reengineered business process). Client applications provide targeted functionality to the user community.

The functions of the existing database management system and the data architecture of the existing database must be reverse engineered as a precursor to the redesign of the database foundation layer. In some cases a new data model is created. In every case, the client-server database is reengineered to ensure that transactions are executed in a consistent manner, that all updates are performed only by authorized users, that core business rules are enforced (e.g., before a vendor record is deleted, the server ensures that no related accounts payable, contracts, or communications exist for that vendor), that queries can be accommodated efficiently, and that full archiving capability has been established.

The business rules layer represents software resident at both the client and the server. This software performs control and coordination tasks to ensure that transactions and queries between the client application and the database conform to the established business process. The client applications layer implements business functions that are required by specific groups of end users. In many instances, a mainframe application is segmented into a number of smaller, reengineered desktop applications. Communication among the desktop applications (when necessary) is controlled by the business rules layer.

7.9 THE ECONOMICS OF REENGINEERING

In a perfect world, every unmaintainable program would be retired immediately; to be replaced by high-quality, reengineered applications developed using modern software engineering practices. But we live in a world of limited resources. Reengineering drains resources that can be used for other business purposes. Therefore, before an organization attempts to reengineer an existing application, it should perform a cost-benefit analysis.

A cost-benefit analysis model for reengineering has been proposed by Sneed. Nine parameters are defined:

P1 = current annual maintenance cost for an application
P2 = current annual operations cost for an application
P3 = current annual business value of an application
P4 = predicted annual maintenance cost after reengineering
P5 = predicted annual operations cost after reengineering
P6 = predicted annual business value after reengineering
P7 = estimated reengineering costs
P8 = estimated reengineering calendar time

P9 = reengineering risk factor (P9 1.0 is nominal)

L = expected life of the system

The cost associated with continuing maintenance of a candidate application (i.e., reengineering is not performed) can be defined as

C_{maint} = [P3 (P1 + P2)] x L - equation 1

The costs associated with reengineering are defined using the following relationship:

C_{reeng} = P6 - (P4 + P5) x (L - P8) - (P7 x P9) - equation 2

Using the costs presented in Equations 1 and 2, the overall benefit of reengineering can be computed as

Cost benefit = C_{reeng} - C_{maint} - equation 3

The cost-benefit analysis presented in these equations can be performed for all high priority applications identified during inventory analysis. Those applications that show the highest cost-benefit can be targeted for reengineering, while work on others can be postponed until resources are available.

7.10 CHECK YOUR PROGRESS

1. What is the purpose of Software Maintenance?
2. What are the types of software maintenance?
3. What are the objectives of Software Reengineering?
4. Name of the Software Reengineering Activities.

Answers to check your progress:

1. The purpose of Software Maintenance:
    - Correct errors
    - Change in user requirement with time
    - Changing hardware/software requirements
    - To improve system efficiency
    - To optimize the code to run faster
    - To modify the components
    - To reduce any unwanted side effects.
2. Corrective maintenance
    - Adaptive maintenance
    - Perfective maintenance

- Preventive maintenance

3. Objectives of Re-engineering:
   - To describe a cost-effective option for system evolution.
   - To describe the activities involved in the software maintenance process.
   - To distinguish between software and data re-engineering and to explain the problems of data re-engineering.
4. The Software Reengineering Activities are:
   - Inventory Analysis
   - Document reconstructing
   - Reverse Engineering
   - Code Reconstructing
   - Data Restructuring
   - Forward Engineering

7.11 SUMMARY

Software maintenance and support are ongoing activities that occur throughout the life cycle of an application. During these activities, defects are corrected, applications are adapted to a changing operational or business environment, enhancements are implemented at the request of stakeholders, and users are supported as they integrate an application into their personal or business workflow.

Reengineering occurs at two different levels of abstraction. At the business level, reengineering focuses on the business process with the intent of making changes to improve competitiveness in some area of the business. At the software level, reengineering examines information systems and applications with the intent of restructuring or reconstructing them so that they exhibit higher quality.

Software reengineering encompasses a series of activities that include inventory analysis, document restructuring, reverse engineering, program and data restructuring, and forward engineering. The intent of these activities is to create versions of existing programs that exhibit higher quality and better maintainability—programs that will be viable well into the twenty-first century.

The cost-benefit of reengineering can be determined quantitatively. The cost of the status quo, that is, the cost associated with ongoing support and maintenance of an existing

application, is compared to the projected costs of reengineering and the resultant reduction in maintenance and support costs. In almost every case in which a program has a long life and currently exhibits poor maintainability or supportability, reengineering represents a cost-effective business strategy.

7.12 KEYWORDS

- **Corrective maintenance:** It aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation, etc.
- **Adaptive maintenance:** It is the modification of software to keep it usable after a change to its operating environment.
- **Perfective software maintenance**: It aims to adjust software by adding new features as necessary and removing features that are irrelevant or not effective in the given software.
- **Preventative Software Maintenance**: It helps to make changes and adaptations to your software so that it can work for a longer period of time.

7.13 QUESTIONS FOR SELF STUDY

1. Explain the Role of Maintenance and Re-engineering in Software Project Management.
2. Describe Software Reengineering with a neat diagram
3. Write a note on Business Process Engineering, challenges, advantages and disadvantages.
4. Explain Software Re-engineering activities with a neat diagram.
5. What are the different types of Software Reverse Engineering? Explain each one of the briefly.
6. What are the differences between restructuring and Forward Engineering?

7.14 REFERENCES

1. https://www.javatpoint.com
2. https://www.geeksforgeeks.org
3. https://www.techtarget.com/

UNIT-8: PROJECT PROCUREMENT MANAGEMENT

STRUCTURE

8.0 OBJECTIVES

After studying this unit, you will be able to:

- Understand the different between business requirements and operational requirements.
- Manage the Procurement Process and the Supply Base Efficiently and Effectively.
- Develop strong relationships with other groups within the organization.
- Support organizational Goals and Objective.

8.1 INTRODUCTION

In this unit, we are going to discuss about project procurement management. The success of many IT projects that use outside resources is often due to good project procurement management. Project procurement management includes the processes required to acquire goods and services for a project from outside the performing organization. Organizations can be either the buyer or seller of products or services under a contract or other agreement.

8.2 INTRODUCTION TO PROJECT PROCUREMENT MANAGEMENT

Project procurement management is the business process by which projects are contracted, outsourced, and finished while the necessary products to complete the projects are selected, coordinated, and maintained. This management concept also applies to the collaborative relationships formed with outside suppliers to obtain, purchase, or maintain a stock of particular goods or services. Most project procurement management relationships are based on a contract to ensure that the required goods or services are received in a timely manner and in the proper condition to meet the standards of the purchasing company. Cost, time, and quality are three main concerns of project procurement management.

Project procurement management is an important part of business that is often controlled by the accounting and purchasing departments of an organization. When this process is employed, strong and structured relationships are created to benefit the purchasing organization most directly; a number of vendors typically bid to receive a contract with the purchasing company and a project procurement manager must then select the most beneficial supplier to form a business relationship with. Project procurement management is vitally important because it encourages fair representation of both the supplier and the purchaser in addition to making considerations for purchase planning, the determination of standards, supplier research, price negotiation, and inventory control.

8.3 PLANNING PURCHASE AND ACQUISITIONS

The purpose of the plan purchases and acquisitions process is to identify which project needs can best be met by purchasing and acquiring products, services, or results outside the project organization. This process involves consideration of whether, how, what, how much, and when to acquire. When the project obtains products, services, and results required for project performance from outside the performing organization, the processes from plan purchases and acquisitions through contract closure are performed for each item to be purchased or acquired.

The plan purchases and acquisitions process encompasses the consideration of potential vendors, particularly if the buyer likes to exercise some degree of influence or control over contracting decisions. Emphasis should also be given to who is responsible for obtaining or holding any relevant permits and professional licenses that may be required by legislation, regulation, or organizational policy in executing the project.

The project schedule is a key input to create plan purchases and acquisitions process. Decisions made in developing the procurement management plan can also influence the project schedule and are integrated with schedule development, activity resource estimating and make or buy decisions.

8.3.1 ORGANIZATIONAL PROCESS ASSETS

Existing formal and informal procurement-related policies, procedures, guidelines, and management systems that are considered in developing the procurement management plan and selecting the contract types to be used are provided by organizational process assets. In some application areas, organizations also have established a multi-tier supplier system of selected and pre-qualified sellers to reduce the number of direct sellers to the organization and establish an extended supply-chain.

Project scope statement

Project boundaries, requirements, constraints and assumptions related to the project scope are described in the project scope statement. Constraints are specific factors that can limit both the buyers and sellers options. One of the most common constraints for many projects is availability of resources. Other constraints can involve required delivery dates, available skilled resources and organizational policies. Assumptions are factors that will be considered to be true and which can include health, safety, security, performance, environmental, insurance, intellectual property rights, equal employment opportunity, licenses and permits.

The project scope statement provides the list of deliverables and acceptance criteria for the project and its products, services and results. Consideration is given to all such factors that may need to be included in the procurement documentation and flowed down within a contract to vendors.

Important information about any technical issues or concerns related to the products, services, and results of the project that are considered during the plan purchases and acquisitions process is provided by the product scope description component of the project scope statement, whereas the structured and detailed plan for the projects scope is provided by the work breakdown structure (WBS) and WBS dictionary components of the project scope statement.

Work breakdown structure

The WBS establishes the relationship among all the components of the project and the project deliverables.

WBS dictionary

It provides detailed statements of work that provide an identification of the deliverables and a description of the work within each WBS component required to produce each deliverable.

Project management plan

The overall plan for managing the project is provided in the project management plan. It includes subsidiary plans such as a scope management plan. Procurement management plan, quality management plan, and contract management plans provide guidance and direction for procurement management planning. To the extent that other planning outputs are available, those other planning outputs are considered during the plan purchases and acquisition process.

Risk management

It contains risk-related information such as the identified risks, risk owners and risk responses mitigations strategies and contingency plans.

Risk-related contractual agreements

It includes agreements for insurance, services and other items as appropriate that are prepared to specify each party's responsibility for specific risks should they occur.

8.3.2 CONTRACT TYPES

There are various types of contracts for different types of purchases. The type pf contract used and specific contract terms and conditions set the degree of risk being assumed by both the buyer and seller. Contracts generally fall into one of three broad categories.

Fixed price or lump-sum contracts

Fixed price or lump-sum contracts involve a fixed price for a well-defined product. It can also include incentives for meeting or exceeding selected project objectivities such as scheduled targets. The simplest form of a fixed price contract is a purchase order for a specified item to be delivered by a specified data for a specified price.

Cost-reimbursable contracts

Cost-reimbursable contract involves payment (reimbursement) to the seller's actual costs plus a fee typically representing seller profit. Costs are usually classified as direct costs or indirect costs. Direct costs are costs incurred for the exclusive benefit of the project (salaries of full time project staff, etc.). Indirect costs are usually calculated as a percentage of direct costs. Cost-reimbursable contracts often include incentive clauses where if the seller meets or exceeds selected projects objectives, such as schedule targets or total cost, then the seller receives an incentive or bonus payment. Three common types of cost-reimbursable contracts are CPF, CPFF and CPIF.

Cost-Plus-Fee (CPF) or Cost-Plus-Percentage of cost (CPPC)

Sellers receives a fee that varies with the actual cost, calculated as an agreed-upon percentage of the costs and also reimbursed for allowable costs for performing the contract work.

Cost-Plus-Incentive-Fee (CPIF)

The seller receives a predetermined fee and incentive bonus based upon achieving certain performance objective levels set in the contract and is reimbursed for allowable costs for performing the contract work. In some CPIF contracts, if the final costs are less than the expected costs, then both the buyer and seller benefit from the cost savings based upon a pre-negotiated sharing formula.

Time and Material (T&M) contracts

Time and material contracts are a hybrid type of contractual arrangement that contains aspects of both cost-reimbursable and fixed-price type arrangements. These types of contracts resemble cost reimbursable type arrangements in that they are open ended. Buyer does not define the full value of the agreement and the exact quantity of items to be delivered at the time of the contract award. Thus, time and material contracts can grow in contract value as if they were cost-reimbursable type arrangements. Conversely, time and material arrangements can also resemble fixed-price arrangements. For example, the buyer and seller can preset unit rates when both parties agree on the rates for a specific resource category. The requirements (standard or custom product version, performance reporting and cost data submittals) that a buyer imposes on a seller, along with other planning considerations, such as the degree of market competition and degree of risk, will also determine which type of contract will be used. In addition, the seller can consider some of those specific requirements as items that have additional costs. Another consideration relates to the future potential of the product or

service being acquired by the project team. Where such potential can be significant, sellers may be inclined or induced to charge prices that are less than would be the case without such future sale potential. While this can reduce the costs to the project, there are legal ramifications if the buyer promises such potential and is not, in fact, realized.

8.4 PLANNING CONTRACTING – TOOLS AND TECHNIQUES

Standard Forms

Standard contracts, standard descriptions of procurement items, non-disclosure agreements, proposal evaluation criteria checklists, or standardized versions of all parts of the needed bid documents are included in standard forms. Organizations that perform substantial amounts of procurement can have many of these documents standardized. Buyer and seller organizations performing intellectual property transactions ensure that non-disclosure agreements are approved and accepted before disclosing any project specific intellectual property information to other party.

Procurement Documents

These documents are used to seek proposals from perspective sellers. A term such as proposal is generally used when other considerations, such as technical skills or technical approach, are paramount and terms such as bid, tender, or quotation is generally used when the seller selection decision will be based on price (as when buying commercial or standard items). However, the terms are often used interchangeably and care is taken not to make unwarranted assumptions about the implications of the term used. Common names for different types of procurement documents include invitation for bid, request for proposal, request for quotation, tender notice, invitation for negotiation, and contractor initial response.

8.5 REQUESTING SELLER RESPONSES – TOOLS AND TECHNIQUES

Bidder Conferences

Also called as contractor conferences, vendor conferences and pre-bid conferences are meetings with perspective sellers prior to preparation of a bid or proposal and to ensure that all prospective sellers have a clear, common understanding of the technical and contract requirements. Responses to questions can be incorporated into the procurement documents as amendments. All potential sellers are given equal standing during this initial buyer and seller interaction to produce the best bid.

Advertising

Sellers can often be expanded by placing advertisements in general circulation publications such as newspapers or in specialty publications such as professional journals. Some government jurisdictions require public advertising of pending government contracts.

Develop Qualified Sellers List

If information is readily available in the organizational assets qualified or approved sellers' lists can be developed from there also. The project team can also develop its own sources whether or not that data is available. General information is widely available through internet, library directions, relevant local associations, trade catalogues, and similar sources. Detailed information on specific sources requires more extensive effort, such as site visits or contact with previous customers. Procurement documents can also be sent to determine if some or all the prospective sellers have an interest in becoming a qualified potential seller.

Select Sellers

Bids or proposals are received evaluation criteria, as applicable, to select one or more sellers who are both qualified and acceptable as a seller are applied under select sellers process. Many factors can be evaluated in the seller selection decision process.

Primary determinant for an off-the-shelf item can be the price or cost. But if the seller proves unable to deliver the products, services or results in a timely manner, the lowest proposed price may not be the lowest cost.

Proposals are often separated into technical and commercial sections with each evaluated separately. Sometimes management sections are required as part of the proposal and also have to be evaluated.

Critical products services and results to mitigate risks that can be associated with issues such as delivery schedules and quality requirements could require multiple sources. The potentially higher cost associated with such multiple sellers, including any loss of possible quantity discounts and replacement and maintenance issues are considered.

The overall process of requesting responses for sellers and evaluating sellers' responses can be repeated on major procurement items. A short list of qualified sellers can be established based on a preliminary proposal. A more detailed evaluation can then be conducted based on a more detailed and comprehensive proposal that is requested from the sellers on the short list.

8.6 SELECT SELLERS – TOOLS AND TECHNIQUES

Weighting systems

The method for qualifying qualitative data to minimize the effect of personal prejudice on the seller section is called a weighing system. Most such systems involve assigning a numerical weight to each of the evaluation criteria, rating the prospective sellers on each criterion, multiplying the weight by the rating and totaling the resultant products to compute an overall score.

Independent Estimates

The independent estimate is sometimes referred to as a should-cost estimate which the procuring organization can either prepare on its own or have an independent estimate of the costs as a check on proposed pricing for many procurement items. Significant differences from these cost estimates can be an indication that the contract statement of work was not adequate that the prospective seller either misunderstood or failed to respond fully to the contract statement of work or that the marketplace changed.

Screening System

Establishing minimum requirements of performance for one of the evaluation criteria are involved in the screening system that can employ a weighting system and independent estimates. For example, a prospective seller might be required to propose a project manager who had specific qualifications before the remainder of the proposal would be considered. These screening systems are used to provide a weighted ranking from best to worst for all sellers who submitted a proposal.

Contract Negotiation

In order to reach to a mutual agreement prior to signing the contract, structure and requirements of the contract are clarified in contract negotiation. Final contract language reflects all agreements reached. Subjects covered include responsibilities and authorities, applicable terms and law, technical and business management approaches, proprietary rights, contract financing, technical solution, overall schedule payments and price. Contract negotiations conclude with a document that can be signed by both buyer and seller or the contract can be a revised offer by the seller or a counter offer by the buyer. For complex procurement items contract negotiation can be independent process with inputs (e.g. an issues or open items list) and outputs (e.g. documented decisions) of its own. For simple

procurement items, the terms and conditions of the contract can be fixed and non-negotiable and only need to be accepted by the seller.

Even though the project manager and other members of the project management team may be present during negotiations to provide, if needed, any clarification of the project's technical quality and management requirements, the project manager may not be the lead negotiator on the contract.

Seller Rating Systems

The seller performance evaluation documentation generated during the contract administration process for previous sellers is one source of relevant information. Seller rating systems are developed by many organizations and use information such as the seller's past performance, quality ratings, delivery performance and contractual compliance. These rating systems are used in addition to the proposal evaluations screening system to select sellers.

Expert Judgment

Expert judgment is used in evaluating seller proposals. A multi-discipline review team accomplishes the evaluation of proposals with expertise in each of the areas covered by procurement documents and proposed contract. This can include expertise from functional disciplines such as contracts, legal, finance, accounting, engineering, design, research, development, sales and manufacturing.

Proposal Evaluation Techniques

Based on some expert judgment and some form of evaluation criteria, many different techniques can be used to rate and score proposals. Evaluation criteria can involve both objective and subjective components. Evaluation criteria are usually assigned predefined weighting with respect to each other when used for a formalized proposal evaluation. The proposal evaluation then uses inputs from multiple reviews that are obtained during the select sellers' process and any significant differences in scoring are resolved. Using a weighting system that determines the total weighted score for each proposal, an overall assessment and comparison of all proposals can then be developed. These proposal evaluation techniques also can employ a screening system and use data from a seller rating system.

8.7 ADMINISTERING AND CLOSING THE CONTRACT – TOOLS AND TECHNIQUES

A contract is a legal relationship subject to remedy in the courts that is awarded to each selected seller. It can be in the form of a complex document or a simple purchase order. Regardless of the documents complexity, a contract is a mutually binding legal agreement that obligates the seller to provide the specified products, services, or results and obligates the buyer to pay seller.

The major components in a contract document generally include section headings, statement of work, schedule period of performance bonds, subcontractor approval, change request handling and a termination and disputes mechanism.

Contract Management Plan

Each contract management plan is a subset of the project management plan. A plan to administer the contract for significant purchases or acquisitions is prepared based upon the specific buyer, specified items within the contract, such as documentation and delivery, and performance requirements that the buyers and sellers must meet. The plan covers the contract administration activities throughout the life of the contact.

Resource Availability

The quantity and availability of resources and those dates on which each specific resource can be active or idle are documented.

Procurement Management Plan

It is updated to reflect any approved change requests that are procurement management.

Tools and techniques of contract administration

- Contract change control system
- Buyer-conducted performance review
- Inspections and audits
- Performance reporting
- Payment system
- Claims administration
- Records management system
- Information technology

Contract Closure

Contract addresses each contact applicable to the project or a project phase. It supports the close project process as it involves verification that all work and deliverables were acceptable. The contract closure process also involves administrative activities such as updating records to reflect final results and archiving such information for future use. In multi-phase projects the team of a contract may only be applicable to that phase of the project. Unresolved claims may be subject to litigation after contract closure.

Early termination of a contract can result from a mutual agreement of the parties or from the default of one of the parties. The rights and responsibilities of the parties in the event of an early termination are contained in a terminations clause of the contract based upon those contracts or a portion of the project for clause or convenience at any time.

The buyer may have to compensate the seller for seller's preparations and for any completed and accepted work related to the terminated part of the contract as per those contract terms and conditions.

8.8 CHECK YOUR PROGRESS

1. What is project procurement management?
2. Name the Organizational Process assets.
3. What are Procurement documents?
4. Which are the tools and techniques of contract administration?

Answers to check your progress:

1. Project procurement management is the business process by which projects are contracted, outsourced, and finished while the necessary products to complete the projects are selected, coordinated, and maintained. This management concept also applies to the collaborative relationships formed with outside suppliers to obtain, purchase, or maintain a stock of particular goods or services. Most project procurement management relationships are based on a contract to ensure that the required goods or services are received in a timely manner and in the proper condition to meet the standards of the purchasing company. Cost, time, and quality are three main concerns of project procurement management.
2. Project Scope Statement
    - Work Breakdown Structure

- WBS Dictionary
- Project Management Plan
- Risk Management
- Risk-related Contractual Agreements

3. These documents are used to seek proposals from perspective sellers. A term such as proposal is generally used when other considerations, such as technical skills or technical approach, are paramount and terms such as bid, tender, or quotation is generally used when the seller selection decision will be based on price (as when buying commercial or standard items). However, the terms are often used interchangeably and care is taken not to make unwarranted assumptions about the implications of the term used. Common names for different types of procurement documents include invitation for bid, request for proposal, request for quotation, tender notice, invitation for negotiation, and contractor initial response.
4. Contract change control system
    - Buyer-conducted performance review
    - Inspections and audits
    - Performance reporting
    - Payment system
    - Claims administration
    - Records management system
    - Information technology

8.9 SUMMARY

The success of many IT projects that use outside resources is often due to good project procurement management. Project procurement management includes the processes required to acquire goods and services for a project from outside the performing organization. Organizations can be either the buyer or seller of products or services under a contract or other agreement. There are four main processes in project procurement management:

a. Planning procurement management involves determining what to procure and when and how to do it. In procurement planning, one must decide what to outsource, determine the type of contract, and describe the work for potential sellers. Sellers are providers, contractors, or suppliers who provide goods and services to other organizations. Outputs of this process include a procurement management plan, procurement statements of work,

procurement documents, source selection criteria, make-or-buy decisions, change requests, and project documents updates.

b. Conducting procurements involves obtaining seller responses, selecting sellers, and awarding contracts. Outputs include selected sellers, agreements, resource calendars, change requests, and updates to the project management plan and other project documents.
c. Controlling procurements involves managing relationships with sellers, monitoring contract performance, and making changes as needed. The main outputs of this process include work performance information, change requests, and updates to the project management plan, project documents, and organizational process assets.
d. Closing procurements involves completion and settlement of each contract or agreement, including resolution of any open items. Outputs include closed procurements and organizational process assets updates.

8.10 KEYWORDS

- **Project scope:** It is the part of project planning that involves determining and documenting a list of specific project goals, deliverables, tasks, costs and deadlines.
- **Bidding**: It is used in business all the time to determine the supplier chosen by the Project Management, Engineering, and Procurement.
- **Software Development Contract:** It is an agreement entered into between a company and a software developer where the company mentions their concepts and requirements.
- **Procurement:** This is when you need to purchase, rent or contract with some external resource to meet your project goal.

8.11 QUESTIONS FOR SELF STUDY

1. Briefly explain the role of Project Procurement Planning in SPM.
2. Name of the different types of contract and explain each one briefly.
3. Discuss the different tools and techniques used for selecting sellers.
4. How to perform administering and closing of the contract with different tools and techniques?

8.12 REFERENCES

1. https://study.com/learn/lesson/project-procurement-management-plan-process.html
2. http://aspalliance.com/1149_Understanding_Project_Procurement_Management.all
3. https://wachemo-elearning.net/courses/31778/lessons/chapter-nine-project-procurement-management/topic/9-2-importance-of-project-procurement-management/

Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA IT Specialization

IV Semester

MBSC-4.1G Software Project Management

Block 3

PREFACE

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products, . . . the list is almost endless. Software is virtually inescapable in a modern world. And as we move into the twenty-first century, it will become the driver for new advances in everything from elementary education to genetic engineering.

When a computer software succeeds—when it meets the needs of the people who use it, when it performs flawlessly over a long period of time, when it is easy to modify and even easier to use—it can and does change things for the better. But when software fails—when its users are dissatisfied, when it is error prone, when it is difficult to change and even harder to use—bad things can and do happen. We all want to build software that makes things better, avoiding the bad things that lurk in the shadow of failed efforts. To succeed, we need discipline when software is designed and built. We need an engineering approach.

The whole material is organized into four modules each with four units. Each unit lists the objectives of the study along with the relevant questions, illustrations and suggested reading to better understand the concepts.

Wish you happy reading!!!

KARNATAKA STATE OPEN UNIVERSITY

MUKTHAGANGOTRI, MYSURU-06

Dept. of Studies and Research in Management

MBA IT Specialization

IV Semester

MBSC-4.1G Software Project Management

BLOCK 3: SOFTWARE PRODUCT / PROJECT MANAGEMENT

BLOCK 3 INTRODUCTION

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products, etc. Software will become the driver for new advances in everything from elementary education to genetic engineering.

In this block, we are discussing about the art and discipline of software project management like software project management in which software projects planned, implemented, monitored and controlled.

This block consists of four units and is organized as follows:

Unit 9: Product Metrics: Framework for product metrics, Metrics for Requirement Model, Metrics for Design Model, Design Metrics for WebApps, Metrics for Source Code, Metrics for Testing and Metrics for Maintenance

Unit 10: User – Interface Design: The Golden Rules, User Interface Analysis and Design, Interface Analysis, Interface Design Steps, WebApp Interface Design, Design Evaluation

Unit 11: Project Management concepts: The Management Spectrum, People, The Product, The Process, The Project, W5HH Principle, Critical Practices

Unit 12: Estimation for Software Projects: Observations on Estimation, The Project Planning Process, Software Scope and Feasibility, Resources, Software Project Estimation, Decomposition Techniques, Empirical Estimation Models, Estimation for Object Oriented Projects, Specialized Estimation Techniques

UNIT-9: PRODUCT METRICS

STRUCTURE

9.0 OBJECTIVES

After studying this unit, you will be able to:

- Explain about software product
- Analyze Project management
- Discuss software product metrics and their requirements
- Evaluate software metrics
- Describe metrics for testing

9.1 INTRODUCTION

In this unit, we are going to discuss about product metrics. Software metrics provide a quantitative way to assess the quality of internal product attributes, thereby enabling the software engineer to assess quality before the product is built. This unit introduces concepts such as framework of product metrics, Metrics for Requirement Model, Metrics for Design

Model and web applications. Also, Metrics for Source Code, Metrics for Testing , Metrics for Maintenance are discussed in detail.

9.2 INTRODUCTION TO PRODUCT METRICS

A key element of any engineering process is measurement. Measures are used to better understand the attributes of the models that we create and to assess the quality of the engineered products or systems that is built.

Software measurement is concerned with deriving a numeric value or profile for an attribute of a software component, system, or process. By comparing these values to each other and to the standards that apply across an organization, based on these measures we can draw conclusions about the quality of software, or assess the effectiveness of software processes, tools, and methods.

Software metric is a characteristic of a software system, system documentation, or development process that can be objectively measured. Examples of metrics like the size of a product in lines of code; the Fog index, which is a measure of the readability of a passage of written text; the number of reported faults in a delivered software product; and the number of person-days required to develop a system component.

Software metrics domain is classified into process, project, and product metrics. **Product metrics** are private to an individual are often combined to develop project metrics that are public to a software team. Project metrics are then consolidated to create process metrics that are public to the software organization as a whole. A software engineer needs objective criteria to guide the design of data, architecture, interfaces, and components. The tester needs quantitative guidance that will help to generate and select the test cases to reach targets. Technical metrics provide a basis from which analysis, design, coding, and testing can be conducted more objectively and assessed more quantitatively.

Technical metrics provide us with a systematic way to evaluate quality based on a set of rules. These rules provide the software engineer with on-the-spot, rather than after-the-fact insight. This enables the engineer to discover and correct potential problems before they become catastrophic defects.

9.3 FRAMEWORK FOR PRODUCT METRICS

Here we can discuss the measures that can be used to estimate the quality of the product as it is being engineered. These measures of internal product attributes provide the software engineer with a real-time indication of the efficacy of the analysis, design, and code models; the effectiveness of test cases; and the overall quality of the software to be built.

As we discussed earlier, measurement assigns numbers or symbols to attributes of entities in the real world. To perform this, a measurement model encompassing a consistent set of rules is required. It is required to establish a fundamental framework and a set of basic principles for the evaluation of technical metrics for software.

9.3.1 THE CHALLENGE OF TECHNICAL METRICS

Although there are other measures have been proposed, each takes different view of what complexity is and what attributes of a system lead to complexity. Hence it is required to measure and control software complexity. It is not easy to derive a single value metric; it should be possible to develop measures of different internal program attributes like effective modularity, functional independence, and other. These measures and the metrics derived can be used as independent indicators of the quality of analysis and design models. But here again, problems arise that many people argue that technical measurement conducted during the early stages of the software process provides software engineers with a consistent and objective mechanism for assessing quality.

In spite of the intuitive connections between the internal structure of software products or technical metrics and its external product and process attributes, there have actually been very few scientific attempts to establish specific relationships. There are a number of reasons why this is so; the most commonly cited is the impracticality of conducting relevant experiments. Each of the challenge noted here is a cause for caution, Measurement is essential if quality is to be achieved.

9.3.2 MEASUREMENT PRINCIPLES

Measurement process that can be characterized by five activities:

1. ***Formulation****:* The derivation of software measures and metrics those are appropriate for the representation of the software that is being considered.

The principles that can be associated with the formulation of technical metrics are

- The objectives of measurement should be established before data collection begins.

- Each technical metric should be defined in an unambiguous manner.
- Metrics should be derived based on a theory that is valid for the domain of application.
- Metrics should be tailored to best accommodate specific products and processes.

2. *Collection:* The mechanism used to accumulate data required to derive the formulated metrics.

3. Analysis: The computation of metrics and the application of mathematical tools.

4. *Interpretation:* The evaluation of metrics results in an effort to gain insight into the quality of the representation.

5. *Feedback:* Recommendations derived from the interpretation of technical metrics transmitted to the software team.

9.3.3 The Attributes of Effective Software Metrics

Hundreds of metrics have been proposed for computer software, but not all provide practical support to the software engineer. Below are the set of attributes that should be encompassed by effective software metrics.

• ***Simple and computable:*** It should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort or time.

• ***Empirically and intuitively persuasive:*** The metric should satisfy the engineer's intuitive notions about the product attribute under consideration (e.g., a metric that measures module cohesion should increase in value as the level of cohesion increases).

• ***Consistent and objective:*** The metric should always yield results that are unambiguous. An independent third party should be able to derive the same metric value using the same information about the software.

• ***Consistent in its use of units and dimensions:*** The mathematical computation of the metric should use measures that do not lead to strange combinations of units.

• ***Programming language independent:*** Metrics should be based on the analysis model, the design model, or the structure of the program itself. They should not be dependent on the vagaries of programming language syntax or semantics.

• ***An effective mechanism for high-quality feedback:*** That is, the metric should provide a software engineer with information that can lead to a higher quality end product.

9.4 METRICS FOR REQUIREMENT MODEL

Technical work in software engineering begins with the creation of the analysis model. It is at this stage that requirements are derived and that a foundation for design is established. Therefore, technical metrics that provide insight into the quality of the analysis model are desirable. These metrics examine the analysis model with the intent of predicting the size of the resultant system. It is likely that size and design complexity will be directly correlated.

9.4.1 FUNCTION-BASED METRICS

Function point metrics provide a standardized method for measuring the various functions of a software application. It measures the functionality from the user's point of view, that is, on the basis of what the user requests and receives in return. Function point analysis is a standard method for measuring software development from the user's point of view.

The function point metric can be used effectively as a means for predicting the size of a system that will be derived from the analysis model. To illustrate the use of the FP metric, we consider a simple requirement and analysis model representation. Figure 9.1 describes a data flow diagram for a function within the SafeHome software. The function manages user interaction, accepting a user password to activate or deactivate the system, and allows inquiries on the status of security zones and various security sensors. The function displays a series of prompting messages and sends appropriate control signals to various components of the security system. The data flow diagram is evaluated to determine the key measures required for computation of the function point metric:

- Number of user inputs
- Number of user outputs
- Number of user inquiries
- Number of files
- Number of external interfaces

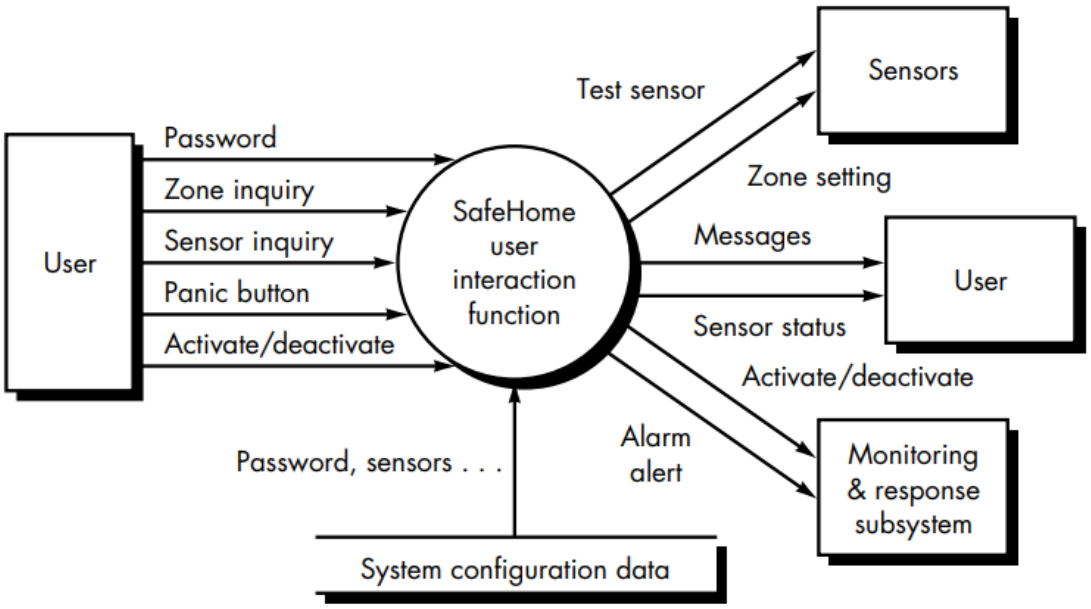


FIGURE 9.1 Part of the requirement analysis model for SafeHome software

			Weighting Factor			
Measurement parameter	**Count**		**Simple**	**Average**	**Complex**	
Number of user inputs	3	×	3	4	6	= 9
Number of user outputs	2	×	4	5	7	= 8
Number of user inquiries	2	×	3	4	6	= 6
Number of files	1	×	7	10	15	= 7
Number of external interfaces	4	×	5	7	10	= 20
Count total						50

FIGURE 9.2 computing function points for a SafeHome function

Three user inputs—password, panic button, and activate/deactivate

Two inquires—zone inquiry and sensor inquiry.

One file - system configuration file.

Two user outputs - messages and sensor status and

Four external interfaces -test sensor, zone setting, activate/deactivate, and alarm alert.

The count total shown in Figure 9.2 must be adjusted using the below equation.

FP = count total x $[0.65 + 0.01 \times \sum(F_i)]$

Where count total is the sum of all FP entries obtained from Figure 9.2 and F_i (i = 1 to 14) are complexity adjustment values. For the purposes of this example, we assume that (F_i) is 46

(moderately complex product). Therefore,

FP = 50 x [0.65 + (0.01 x 46)] = 56

Based on the projected FP value derived from the analysis model, the project team can estimate the overall implemented size of the SafeHome user interaction function. Assume that past data indicates that one FP translates into 60 lines of code (an object oriented language is to be used) and that 12 FPs are produced for each person-month of effort. These historical data provide the project manager with important planning information that is based on the analysis model rather than preliminary estimates. Assume further that past projects have found an average of three errors per function point during analysis and design reviews and four errors per function point during unit and integration testing. These data can help software engineers assess the completeness of their review and testing activities.

9.4.2 THE BANG METRIC

Like the function point metric, the bang metric can be used to develop an indication of the size of the software to be implemented as a consequence of the analysis model. Developed by DeMarco, the bang metric is an implementation independent indication of system size.

To compute the bang metric, the software engineer must evaluate a set of primitives elements of the analysis model that are not further subdivided at the analysis level. Primitive metrics are determined by evaluating the analysis model and developing counts for the following forms:

Functional primitives (FuP): The number of transformations that appear at the lowest level of a data flow diagram.

Data elements (DE): The number of attributes of a data object, data elements are not composite data and appear within the data dictionary.

Objects (OB): The number of data objects.

Relationships (RE): The number of connections between data objects.

States (ST): The number of user observable states in the state transition diagram.

Transitions (TR): The number of state transitions in the state transition diagram.

In addition to these six primitives, additional counts are determined as follows,

Modified manual function primitives (FuPM): Functions, that lies outside the system boundary. These functions are modified to accommodate the new system.

Input data elements (DEI): Those data elements that are input to the system.

Output data elements (DEO): Those data elements that are output from the system.

Retained data elements (DER): Data elements are retained (stored) by the system.

Data tokens (TC_i): The data tokens (data items that are not subdivided within a functional primitive) that exist at the boundary of the i^{th} functional primitive (evaluated for each primitive).

Relationship connections (REi): The relationships that connect the i^{th} object in the data model to other objects.

Software can be allocated to one of two domains function strong or data strong, depending upon the ratio RE/FuP. *Function-strong applications* are often encountered in engineering and scientific applications emphasize the transformation of data and do not generally have complex data structures. *Data-strong applications* are often encountered in information systems applications tend to have complex data models.

RE/FuP < 0.7 implies a function-strong application.

0.8 < RE/FuP < 1.4 implies a hybrid application.

RE/FuP > 1.5 implies a data-strong application.

9.4.3 METRICS FOR SPECIFICATION QUALITY

Davis and his team propose a list of characteristics that can be used to assess the quality of the analysis model and the corresponding requirements specification: specificity (lack of ambiguity), completeness, correctness, understandability, verifiability, internal and external consistency, achievability, concision, traceability, modifiability, precision, and reusability.

Each can be represented using one or more metrics. For instance, we assume that there are n_r requirements in a specification, using the following equation $n_r = n_f + n_{nf}$ where, n_f is the number of functional requirements and n_{nf} is the number of non-functional (e.g., performance) requirements.

To determine the specificity (lack of ambiguity) of requirements, a metric that is based on the consistency of the reviewers' interpretation of each requirement using the equation $Q_1 = n_{ui}/n_r$ where n_{ui} is the number of requirements for which all reviewers had identical interpretations. The closer the value of Q to 1, the lower is the ambiguity of the specification.

The completeness of functional requirements can be determined by computing the ratio $Q2 = n_u/[n_i \times n_s]$ where, n_u is the number of unique function requirements, n_i is the number of inputs (stimuli) defined or implied by the specification, and n_s is the number of states specified. The Q2 ratio measures the percentage of necessary functions that have been specified for a system. However, it does not address nonfunctional requirements. To incorporate these into an overall metric for completeness, we must consider the degree to

which requirements have been validated with Q3 = $n_c/[n_c + n_{nv}]$ where n_c is the number of requirements that have been validated as correct and n_{nv} is the number of requirements that have not yet been validated.

9.5 METRICS FOR DESIGN MODEL

Metrics simply measures quantitative assessment that focuses on countable values most commonly used for comparing and tracking performance of system. Metrics are used in different scenarios like analyzing model, design model, source code, testing, and maintenance. Metrics for design modeling allows developers or software engineers to evaluate or estimate quality of design and include various architecture and component-level designs.

9.5.1 ARCHITECTURAL DESIGN METRICS

Architectural design metrics focus on characteristics of the program architecture with an emphasis on the architectural structure and the effectiveness of modules. These metrics are black box in the sense that they do not require any knowledge of the inner workings of a particular software component.

Metrics by Glass and Card:

In designing a product, it is very important to have efficient management of complexity. Glass and Card are two scientists who have suggested three design complexity measures.

These are describes are follows,

Structural Complexity –

Structural complexity depends upon **fan-out** for modules. It can be defined by, $S(k) = f^2out(k)$ Where f_{out} represents fan_{out} for module k (fan-out means number of modules that are subordinating module k).

Data Complexity –

Data complexity is complexity within interface of internal module. It is size and intricacy of data. For some module k, it can be defined as D(k) = tot_var(k) / [fout(k)+1] where tot_var is total number of input and output variables going to and coming out of module.

System Complexity –

System complexity is combination of structural and data complexity. It can be denoted as

Sy(k) = S(k)+D(k). When, structural data and system complexity get increased then overall architectural complexity also gets increased.

Complexity metrics –

Complexity metrics are used to measure complexity of overall software. The computation of complexity metrics can be done with help of a flow graph. It is sometimes called **cyclomatic complexity**. The cyclomatic complexity is a useful metric to indicate complexity of software system. Without use of complexity metrics, it is very hard and time-consuming to determine complexity in designing products where risk cost emanates. Even continuous complexity analysis makes it difficult for project team and management to solve problem. Measuring Software complexity leads to improve code quality, increase productivity, meet architectural standards, reduce overall cost, increases robustness, etc. To calculate cyclomatic complexity, following equation: Cyclomatic complexity= E - N + 2 where, E is the total number of edges and N is total number of nodes.

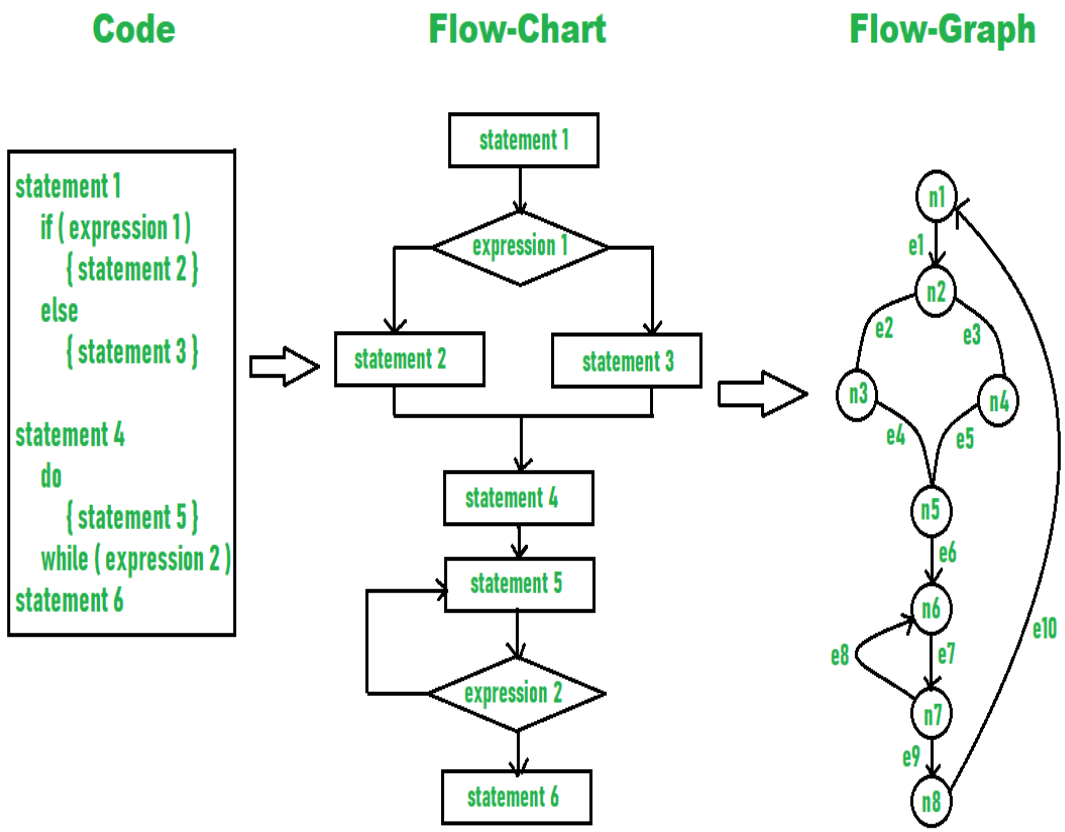


FIGURE 9.3: Flow chart and flow graph for the example code

For example, in the figure 9.3, we can observe the number of edges and number of nodes.
So, the Cyclomatic complexity can be calculated as –

Given,

E = 10, N = 8

So,

Cyclomatic complexity = E - N + 2

= 10 – 8 + 2

= 4

9.5.2 COMPONENT-LEVEL DESIGN METRICS

Component-level design metrics focus on internal characteristics of a software component and include measures of the "three Cs"—module cohesion, coupling, and complexity. These measures can help a software engineer to judge the quality of a component-level design.

COHESION METRICS

Cohesion metrics define a collection of metrics that provide an indication of the cohesiveness of a module. The metrics are defined in terms of five concepts and measures:

- **Data slice:** A data slice is a backward walk through a module that looks for data values that affect the module location at which the walk began. It should be noted that both program slices (which focus on statements and conditions) and data slices can be defined.
- **Data tokens:** The variables defined for a module can be defined as data tokens for the module.
- **Glue tokens:** This set of data tokens lies on one or more data slice.
- **Superglue tokens:** These data tokens are common to every data slice in a module.
- **Stickiness:** The relative stickiness of glue token is directly proportional to the number of data slices that it binds.

All the above cohesion metrics range in value between 0 and 1.

COUPLING METRICS

Module coupling provides an indication of the connectedness of a module to other modules, global data, and the outside environment.

For data and control flow coupling,

di = number of input data parameters

ci = number of input control parameters

do = number of output data parameters

co = number of output control parameters

For global coupling,

gd = number of global variables used as data

gc = number of global variables used as control

For environmental coupling,

w = number of modules called (fan-out)

r = number of modules calling the module under consideration (fan-in)

Using these measures, a module coupling indicator, mc, is defined in the following way:

mc = k/M where k = 1, a proportionality constant 8 and

M = di + (a x ci) + do + (b x co) + gd + (c x gc) + w + r where a = b = c = 2.

The higher the value of mc, the lower is the overall module coupling.

COMPLEXITY METRICS

A variety of software metrics can be computed to determine the complexity of program control flow. Many of these are based on the flow graph. A graph is a representation composed of nodes and links (also called edges). When the links (edges) are directed, the flow graph is a directed graph. Complexity metrics can be used to predict critical information about reliability and maintainability of software systems from automatic analysis of source code. Complexity metrics also provide feedback during the software project to help and control the design activity. During testing and maintenance, they provide detailed information about software modules to help pinpoint areas of potential instability.

9.5.3 INTERFACE DESIGN METRICS

A typical GUI uses layout entities—graphic icons, text, menus, windows, and the like—to assist the user in completing tasks. To accomplish a given task using a GUI, the user must move from one layout entity to the next. The absolute and relative position of each layout entity, the frequency with which it is used, and the "cost" of the transition from one layout entity to the next all contribute to the appropriateness of the interface.

9.6 DEIGN METRICS FOR WEBAPP

The web page metrics are used in measuring various attributes of any website effectively. There are various measures or metrics that can be measured in web testing like finding the audience page views, pages per session, devices they are using, types of traffics like direct, indirect, organic, search, and also site content or site speed, etc.

Measures of web metrics: There are a lot of things that you can measure for growing your website or a business so that there can be more traffic or users available on your website. Some of the best examples that you can measure in your website using the tools available in the market are given below:

Site Traffic: This is the most common metric that every developer or businessman checks for its website growth. It gives us a good indication of your website if it is growing, or declining. It also shows you the various traffic sources like unique visitors or repeats visitors, etc.

Bounce Rate: This is also another important metric when monitoring your website for success. The bounce rate tells you the percentage of visitors who leave your website immediately after clicking on it or after arriving. So, thus it means that if the site has less bounce rate then there will be a great growth of the site otherwise the site is not at a good performance.

Session Duration: The session duration tells us about how much time a user was active on the website. If your website gets a higher number of clicks, then the session duration will also be higher. As per Google Analytics, if the user is inactive on the website for 30 minutes, it assumes that the session is closed.

Site Speed: This metric helps the user to see how much time a page is taking to load. It is calculated as the average page load time.

Page views: This metric show which page of your website has more traffic or more views. If you have the idea that which content of your page is most liked by the user, then you can improve or add more content like that so that you can have more traffic.

Conversion Rate: This metric is calculated as the number of conversions divided by the total number of visitors. It is a very important topic for those that have a sales website as if they know about the conversion rate of their website, then they can increase their sales profit through this metric. For example, if the site has 100 unique users and has sales of 25 users, then the conversion rate will be 25/100 = 25%.

Devices Sources: This metric shows the type of devices (PC, Mobile, Tablet, etc) a user is opening the website. It can help you in optimizing the website for each device or the operating systems they are using.

9.6.1 TYPES OF TRAFFIC SOURCES:

We know that website traffic is the most essential metric that every person checks who has a website as it shows how successful the website is. The traffic sources are divided into 4

categories:

Organic Traffic: It is a type of traffic that comes from various search engines that a user has searched for. This traffic is the most natural source that a website has earned. It can be increased by applying a technique called Search Engine Optimization (SEO).

Direct traffic: It is a type of traffic in which the user directly comes either by direct the full URL or by searching the website name. This traffic consists of the regular visitors of your site.

Social traffic: It is a type of traffic that comes from various traffic sources like Facebook, WhatsApp, Instagram, Twitter, etc.

Referral traffic: It is a type of traffic that comes when your website is linked with some other website or comes if the website link is attached to a blog post or any other method as a means of referral.

There are three types of metrics used in testing a website. They are:

1. Page composition metrics: The composition of a page consists of the arrangement of all visual elements on a webpage like links, words, percentage related to how much size does the page consuming, etc. Some of the common metrics for measuring the various attributes in a page are shown below with their description.

i. Number of words → Checks the total words on a page
ii. Body text words → Checks the words that are body Vs. display text
iii. Embedded links → Checks the links embedded in text on a page
iv. Readability → Checks the reading level of text on a page
v. Navigation percentage → Checks the total portion of a page given to navigation
vi. Page size → Checks the total bytes for the page and images
vii. Animated elements → Checks the animated images and scrolling text
viii. Length of link text → Checks the words in the text for a link

2. Page formatting metrics: There are various metrics that need to be tested in a webpage as it defines how well a website is formatted and how many styles, colors, tables, or screens are used in the website. Some of the common page formatting metrics are shown below with their description.

i. Font styles → Checks the type of fonts used
ii. Types of fonts → Checks the total emphasized text
iii. Number of font sizes → Checks the total font sizes employed
iv. Text clustering → Checks the text areas highlighted with the color

v. Number of colors → Checks the total colors employed
vi. Frames → Use of frames
vii. Number of tables → Checks the total tables added on a web page
viii. Number of screens → Checks the total scrolls required on the screens

3. Page Quality or Assessment Metrics: For measuring the quality of any website or to assess a website, we have to check various things that include the links quality, the speed of loading and downloading, the quality of the images on different browsers, etc. Some of the common metrics are shown below with their description.

i. Information quality → Content appropriateness like tone, etc.
ii. Link quality → Checks the relevance of the links.
iii. Layout quality → Checks the alignment and balance.
iv. Download speed → Checks the time needed for a page to fully load.
v. Image quality → Checks the image appropriateness, size, and resolution.

9.6.2 BENEFITS OF WEB METRICS:

- The metrics help us in determining the direction
- It also creates a focal point
- It helps in making small or large decisions
- It determines the website performance
- It also provides real-time user data

9.6.3 POPULAR TOOLS FOR MEASURING WEB METRICS:

Some of the most popular tools used in measuring the web-metrics are given below:

- Google Analytics
- Adobe Analytics
- Baidu Analytics
- StatCounter
- Ahrefs

9.7 METRICS FOR SOURCE CODE

Halstead's theory of software science is one of the best known and most thoroughly studied composite measures of (software) complexity. Software science proposed the first analytical laws for computer software. Software science assigns quantitative laws to the

development of computer software, using a set of primitive measures that may be derived after code is generated or estimated once design is complete.

These follow:

n1 = the number of distinct operators that appear in a program.

n2 = the number of distinct operands that appear in a program.

N1 = the total number of operator occurrences.

N2 = the total number of operand occurrences.

Halstead uses these primitive measures to develop expressions for the overall program length, potential minimum volume for an algorithm, the actual volume (number of bits required to specify a program), the program level (a measure of software complexity), the language level (a constant for a given language), and other features such as development effort, development time, and even the projected number of faults in the software.

Halstead shows that length N can be estimated using $N = n1* \log_2 n1 + n2 * \log_2 n2$ and program volume may be defined $V = N* \log_2 (n1 + n2)$. It should be noted that V will vary with programming language and represents the volume of information (in bits) required to specify a program.

Theoretically, a minimum volume must exist for a particular algorithm. Halstead defines a volume ratio L as the ratio of volume of the most compact form of a program to the volume of the actual program. In actuality, L must always be less than 1.

In terms of primitive measures, the volume ratio may be expressed as.

$L = 2/n1 \times n2/N2$

Halstead's work is amenable to experimental verification.

9.8 METRICS FOR TESTING

Testers must rely on analysis, design, and code metrics to guide them in the design and execution of test cases.

Function-based metrics can be used as a predictor for overall testing effort. Various project-level characteristics like testing effort and time, errors uncovered, number of test cases produced for past projects can be collected and correlated with the number of FP produced by a project team. The team can then project expected value of these characteristics for the current project.

The bang metric can provide an indication of the number of test cases required by examining the primitive measures discussed in earlier section. The number of functional primitives (FuP), data elements (DE), objects (OB), relationships (RE), states (ST), and transitions (TR) can be used to project the number and types of black-box and white-box tests for the software. For example, the number of tests associated with the human/computer interface can be estimated by

(1) Examining the number of transitions (TR) contained in the state transition representation of the HCI and evaluating the tests required to exercise each transition;
(2) Examining the number of data objects (OB) that move across the interface, and
(3) The number of data elements that are input or output.

Architectural design metrics provide information on the ease or difficulty associated with integration testing and the need for specialized testing software (e.g., stubs and drivers). Cyclomatic complexity (a component-level design metric) lies at the core of basis path testing. In addition, cyclomatic complexity can be used to target modules as candidates for extensive unit testing. Modules with high cyclomatic complexity are more likely to be error prone than modules whose cyclomatic complexity is lower. For this reason, the tester should expend above average effort to uncover errors in such modules before they are integrated in a system. Testing effort can also be estimated using metrics derived from Halstead measures. Using the definitions for program volume, V, and program level, PL, software science effort, e, can be computed as

$PL = 1/[(n1/2)\bullet(N2/n2)]$

$e = V/PL$

Testing metrics fall into two broad categories:

(1) Metrics that attempt to predict the likely number of tests required at various testing levels and
(2) Metrics that focus on test coverage for a given component.

The percentage of overall testing effort to be allocated to a module k can be estimated using the following relationship:

Percentage of testing effort (k) = $e(k)/ \sum e(i)$ where e(k) is computed for module k using the above equation and the summation is the sum of software science effort across all modules of the system.

As test cases are conducted, three different measures provide an indication of testing completeness. A measure of the breath of testing provides an indication of how many requirements (of the total number of requirements) have been tested. This provides an indication

of the completeness of the test plan. Depth of testing is a measure of the percentage of independent basis paths covered by testing versus the total number of basis paths in the program. A reasonably accurate estimate of the number of basis paths can be computed by adding the cyclomatic complexity of all program modules. Finally, as tests are conducted and error data are collected, fault profiles may be used to rank and categorize errors uncovered. Priority indicates the severity of the problem. Fault categories provide a description of an error so that statistical error analysis can be conducted.

9.9 METRICS FOR MAINTENANCE

All of the software metrics introduced in this unit can be used for the development of new software and the maintenance of existing software. However, metrics designed explicitly for maintenance activities have been proposed. A Software Maturity Index (SMI) that provides an indication of the stability of a software product (based on changes that occur for each release of the product). The following information is determined:

MT = the number of modules in the current release

Fc = the number of modules in the current release that have been changed

Fa = the number of modules in the current release that have been added

Fd = the number of modules from the preceding release that were deleted in the current release.

The software maturity index is computed in the following manner:

SMI = [MT (Fa + Fc + Fd)]/MT

SMI may also be used as metric for planning software maintenance activities. The mean time to produce a release of a software product can be correlated with SMI and empirical models for maintenance effort can be developed.

9.10 CHECK YOUR PROGRESS

1. Product metrics that are computed from data collected from.
    (a) The requirements and design models
    (b) Source code
    (c) Test cases
    (d) All of the mentioned above
2. ---------- provide information on the ease or difficulty associated with integration testing.
    (a) Product metrics

(b) Work products

(c) Architectural design metrics

(d) All of the mentioned above

3. ---------- metric can be used effectively as a means for measuring the functionality delivered by a system.

(a) Function point

(b) Product metric

(c) Both a and b

(d) None of the mentioned above

4. Software engineers use ---------- to help them build higher quality software.

(a) Measurable attributes

(b) Work products

(c) Product metrics

(d) All the mentioned above

5. Architectural Design Metrics are ---------- in nature.

(a) Black Box

(b) White Box

(c) Gray Box

(d) Green Box

6. Which of the following metric does not depend on the programming language used?

(A) Line of code

(B) Function count

(C) Member of token

(D) All of the above

7. While estimating the cost of software, Lines of Code (LOC) and Function Points (FP) are used to measure which one of the following?

(A) Length of code

(B) Size of software

(C) Functionality of software

(D) None of the above

8. SMI stands for.

(a) Software mature Indicator

(b) Software Maturity Index

(c) Software Mature Index

(d) Software maturity Indicator

9. Measurement assigns numbers or symbols to attributes of entities in the real world? True/ False.
10. Cyclomatic complexity can be calculated using ----------
11. Define software metrics.

Answers to check your progress:

1. All of the mentioned above
2. Architectural design metrics
3. Function point
4. Product metrics
5. Black Box
6. Function count
7. Size of software
8. Software Maturity Index
9. True
10. Cyclomatic complexity= E - N + 2
11. Software metric is a characteristic of a software system, system documentation, or development process that can be objectively measured.

9.11 SUMMARY

Software metrics provide a quantitative way to assess the quality of internal product attributes, thereby enabling the software engineer to assess quality before the product is built. Metrics provide the insight necessary to create effective analysis and design models, solid code, and thorough tests. To be useful in a real world context, software metric must be simple and computable, persuasive, consistent, and objective. It should be programming language independent and provide effective feedback to the software engineer.

Metrics for the requirement analysis model focus on function, data, and behavior, the three components of the analysis model. The function point and the bang metric each provide a quantitative means for evaluating the analysis model. Metrics for design consider architecture, component-level design, and interface design issues. Architectural design metrics consider the structural aspects of the design model. Component-level design metrics provide an indication of module quality by establishing indirect measures for cohesion,

coupling, and complexity. Interface design metrics provide an indication of layout appropriateness for a GUI. Software science provides an intriguing set of metrics at the source code level. Using the number of operators and operands present in the code, software science provides a variety of metrics that can be used to assess program quality. Few technical metrics have been proposed for direct use in software testing and maintenance. However, many other technical metrics can be used to guide the testing process and as a mechanism for assessing the maintainability of a computer program.

9.12 KEYWORDS

- **Software metric**- is a characteristic of a software system, system documentation, or development process that can be objectively measured.
- **Product metrics**- are software product measures at any stage of their development, from requirements to established systems.
- **Fan-out-** means number of modules that are subordinating module (say k).
- **Cohesion metrics-** is a collection of metrics that provide an indication of the cohesiveness of a module.
- **Data tokens-** are the variables defined for a module.
- **Superglue tokens-** are common data tokens to every data slice in a module.
- **Bounce rate-** is the percentage of visitors who leave your website immediately after clicking on it or after arriving.

9.13 QUESTIONS FOR SELF STUDY

1. What is software metrics? Explain its classification.
2. What are the challenges of technical metrics?
3. What are the activities of measurement process?
4. Explain the attributes of effective software metrics.
5. Define the following terms: (a) Product metrics (b) Glue token (C) Superglue token (d) Organic traffic (e) Direct traffic (f) Referral traffic (g) Bounce rate
6. What are functions based metrics? Explain.
7. Give a brief explanation on the metrics of design.
8. List the benefits of web metrics.
9. What are the different types of metrics used in testing a website? Explain.

10. Describe metrics for testing.

9.14 REFERENCES

1. Software Engineering, A Practitioner's Approach – 7th Edition, Roger S.Pressman
2. Software Project Management in Practice – Pankaj Jalote
3. Software Engineering: Pearson New International Edition – Ian Sommerville, 2013.

UNIT-10: USER – INTERFACE DESIGN

STRUCTURE

10.0 OBJECTIVES

After studying this unit, you will be able to:

- Understand the importance of user interface
- Know the types of Interfaces
- Identify the Steps of User Interface Design and analysis
- Gain the knowledge about WebApp interface development
- Able to evaluate the design process

10.1 INTRODUCTION

In this unit, we are going to discuss about user interface design. The user interface is arguably the most important element of a computer-based system or product. If the interface is poorly designed, the user's ability to tap the computational power of an application may be severely hindered. In fact, a weak interface may cause to fail the well designed and implemented application.

10.2 INTRODUCTION TO USER INTERFACE

User Interface (UI) is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc. UI provides fundamental platform for human-computer interaction. UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both. The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interface screens

User interface is broadly classified into two categories:

1. Command Line Interface (CLI)
2. Graphical User Interface (GUI)

Command Line Interface: Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use. A command is a text-based reference to the set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that made it easy for the user to operate. CLI uses less amount of computer resource as compared to GUI.

Graphical User Interface: Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software. GUI is more resource consuming when compared to CLI. With advance technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

10.3 USER INTERFACE DESIGN ACTIVITIES

There are a number of activities to be performed while designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation like Waterfall, Iterative or Spiral Model. Figure 10.1 shows the phased involved in GUI design.

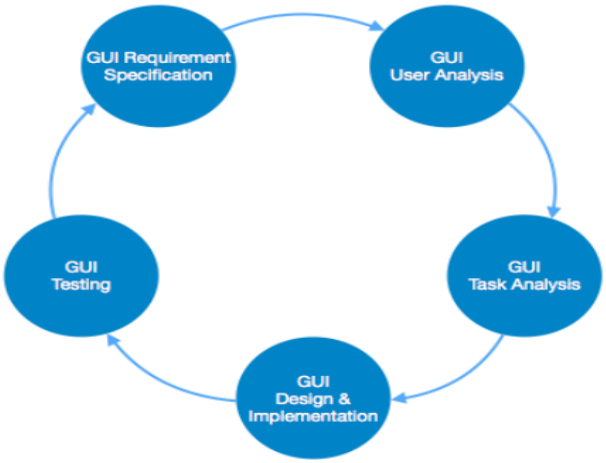


Figure 10.1 Phases of user interface design

- **GUI Requirement Gathering -** The designers wish to have list of all functional and non-functional requirements of GUI. This can be collected from user and the existing software solution.
- **User Analysis -** The designer study about the user, who is going to use the software GUI. The target audience matters, as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.
- **Task Analysis** - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.
- **GUI Design & implementation** - After having information about requirements, tasks and user environment, designers start to design the GUI and implements into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.
- **Testing** - GUI testing can be done in many ways. Organization can have in-house

inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.

10.4 USER INTERFACE DESIGN PROCESS AND ANALYSIS

The overall process for analyzing and designing a user interface begins with the creation of different models of system function (as perceived from the outside). Tools are used to prototype and ultimately implement the design model, and the result is evaluated by end users for quality.

Four different models come into play when a user interface is to be analyzed and designed.

- A human engineer (or the software engineer) establishes a user model,
- The software engineer creates a design model,
- The end user develops a mental image that is often called the user's mental model or the system perception,
- The implementers of the system create an implementation model.

The user model establishes the profile of end users of the system.

To build an effective user interface- all design should begin with an understanding of the intended users, including profiles of their age, gender, physical abilities, education, cultural or ethnic background, motivation, goals and personalities. In addition, users can be categorized as:

a) Novices- No syntactic knowledge of the system and little semantic knowledge of the application or computer usage in general.

b) Knowledgeable, intermittent users- Reasonable semantic knowledge of the application but relatively low recall of syntactic information necessary to use the interface.

c) Knowledgeable, frequent users- Good semantic and syntactic knowledge that often leads to the "power-user syndrome"; that is, individuals who look for shortcuts and abbreviated modes of interaction.

The user's mental model (system perception): is the image of the system that end users carry in their heads. A user who understands word processors fully but has worked with the specific word processor only once might actually be able to provide a more complete description of its function than the novice who has spent weeks trying to learn the system.

The implementation model: combines the outward manifestation of the computer-based

system (the look and feel of the interface), coupled with all supporting information (books, manuals, videotapes, help files) that describes interface syntax and semantics. When the implementation model and the user's mental model are coincident, users generally feel comfortable with the software and use it effectively. To accomplish this "melding" of the models, the design model must have been developed to accommodate the information contained in the user model, and the implementation model must accurately reflect syntactic and semantic information about the interface.

The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities. Figure 10.2 shows the frame work activities.

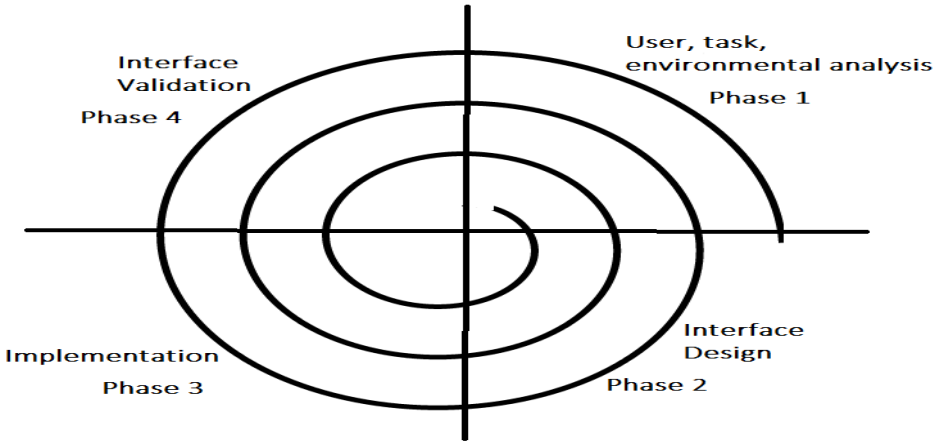


Figure 10.2 User interface framework

User, task, environmental analysis, and modeling: Initially, the focus is based on the profile of users, who will interact with the system, i.e. understanding, skill and knowledge, type of user, etc. Based on the user's profile users are made into categories. From each category requirements are gathered. Depending on the requirements developer understands how to develop the interface. Once all the requirements are gathered a detailed analysis is conducted. In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated. The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:

- Where will the interface be located physically?
- Will the user be sitting, standing, or performing other tasks unrelated to the interface?
- Does the interface hardware accommodate space, light, or noise constraints?
- Are there special human factors considerations driven by environmental factors?

Interface Design: The goal of this phase is to define the set of interface objects and actions i.e. Control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system. Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task. Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.

Interface construction and implementation: The implementation activity begins with the creation of prototype model that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.

Interface Validation: This phase focuses on testing the interface. The interface should be in such a way that it should be able to perform tasks correctly and it should be able to handle a variety of tasks. It should achieve all the user's requirements. It should be easy to use and learn. Users should accept the interface as a useful one in their work.

10.5 GOLDEN RULES

The following are the golden rules stated by Theo Mandel that must be followed during the design of the interface.

1. Place the user in control.
2. Reduce the user's memory load.
3. Make the interface consistent

Place the user in control:

- *Define the interaction modes:* In such a way that does not force the user into unnecessary or undesired actions: The user should be able to easily enter and exit the mode with little or no effort.
- *Provide for flexible interaction:* Different people will use different interaction mechanisms, some might use keyboard commands, some might use mouse, some might use touch screen, etc., and hence all interaction mechanisms should be provided.

- *Allow user interaction to be interruptible and undoable:* When a user is doing a sequence of actions the user must be able to interrupt the sequence to do some other work without losing the work that had been done. The user should also be able to do undo operation.
- *Streamline interaction as skill level advances and allow the interaction to be customized:* Advanced or highly skilled user should be provided a chance to customize the interface as user wants which allows different interaction mechanisms so that user doesn't feel bored while using the same interaction mechanism.
- *Hide technical internals from casual users:* The user should not be aware of the internal technical details of the system. He should interact with the interface just to do his work.
- *Design for direct interaction with objects that appear on screen:* The user should be able to use the objects and manipulate the objects that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.

Reduce the user's memory load:

- *Reduce demand on short-term memory:* Design of short-term memory is significant when users are involved in complex tasks. So the interface should be designed in a way to reduce the remembering of previously done actions, given inputs and results.
- *Establish meaningful defaults:* Always initial set of defaults should be provided to the average user, if a user needs to add some new features then he should be able to add the required features.
- *Define shortcuts that are intuitive:* Mnemonics should be used by the user. **Mnemonics** means the keyboard shortcuts to do some action on the screen.
- *The visual layout of the interface should be based on a real-world metaphor:* Anything you represent on a screen if it is a metaphor for real-world entity then users would easily understand.
- *Disclose information in a progressive fashion:* The interface should be organized hierarchically i.e. on the main screen the information about the task, an object or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick.

Make the interface consistent:

- *Allow the user to put the current task into a meaningful context:* Many interfaces have dozens of screens. So it is important to provide indicators consistently so that the user

know about the doing work. The user should also know from which page has navigated to the current page and from the current page where can navigate.

- *Maintain consistency across a family of applications:* The development of some set of applications all should follow and implement the same design, rules so that consistency is maintained among applications.
- If past interactive models have created user expectations do not make changes unless there is a compelling reason.

10.6 INTERFACE ANALYSIS

A key tenet of all software engineering process models is to understand the problem before attempting to design a solution. In the case of user interface design, understanding the problem means understanding

(1) the people (end users) who will interact with the system through the interface

(2) the tasks that end users must perform to do their work

(3) the content that is presented as part of the interface and

(4) the environment in which these tasks will be conducted.

10.6.1 USER ANALYSIS

To get the mental image and the design model to converge is to work to understand the users themselves as well as how these people will use the system. Information from a broad array of sources can be used to accomplish this:

User Interviews: The most direct approach, members of the software team meet with end users to better understand their needs, motivations, work culture, and a myriad of other issues. This can be accomplished in one-on-one meetings or through focus groups.

Sales input: Sales people meet with users on a regular basis and can gather information that will help the software team to categorize users and better understand their requirements.

Marketing input: Market analysis can be invaluable in the definition of market segments and an understanding of how each segment might use the software in subtly different ways.

Support input: Support staff talks with users on a daily basis. They are the most likely source of information on what works and what doesn't, what users like and what they dislike, what features generate questions and what features are easy to use.

The following set of questions will help you to better understand the users of a system:

• Are users trained professionals, technicians, clerical, or manufacturing workers?
• What level of formal education does the average user have?
• Are the users capable of learning from written materials or have they expressed a desire for classroom training?
• Are users expert typists or keyboard phobic?
• What is the age range of the user community?
• Will the users be represented predominately by one gender?
• How are users compensated for the work they perform?

10.6.2 USER ANALYSIS

The goal of task analysis is to answer the following questions:
• What work will the user perform in specific circumstances?
• What tasks and subtasks will be performed as the user does the work?
• What specific problem domain objects will the user manipulate as work is performed?
• What is the sequence of work tasks—the workflow?
• What is the hierarchy of tasks?

Use case: When used as part of task analysis, the use case is developed to show how an end user performs some specific work-related task. In most instances, the use case is written in an informal style (a simple paragraph) in the first-person. This use case provides a basic description of one important work task for the computer-aided design system. From it, you can extract tasks, objects, and the overall flow of the interaction.

Task elaboration: Task analysis for interface design uses an elaborative approach to assist in understanding the human activities the user interface must accommodate. To understand the tasks that must be performed to accomplish the goal of the activity, one should must understand the tasks that people currently perform (when using a manual approach) and then map these into a similar (but not necessarily identical) set of tasks that are implemented in the context of the user interface. Alternatively, one can study an existing specification for a computer-based solution and derive a set of user tasks that will accommodate the user model, the design model, and the system perception. Regardless of the overall approach to task analysis, one must first define and classify tasks.

Object elaboration: Instead of focusing on the tasks that a user must perform, one can examine the use case and other information obtained from the user and extract the meaningful objects. These objects can be categorized into classes. Attributes of each class are defined,

and an evaluation of the actions applied to each object provide a list of operations.

Workflow analysis: This technique allows you to understand how a work process is completed when several people (and roles) are involved.

Hierarchical representation: A process of elaboration occurs as you begin to analyze the interface. Once workflow has been established, a task hierarchy can be defined for each user type. each task identified for the user.

10.6.3 ANALYSIS OF THE DISPLAY CONTENT

For modern applications, display content can range from character-based reports (e.g., a spreadsheet), graphical displays (e.g., a histogram, a 3-D model, a picture of a person), or specialized information (e.g., audio or video files). The analysis modeling techniques identify the output data objects that are produced by an application. These data objects may be

(1) generated by components (unrelated to the interface) in other parts of an application,
(2) acquired from data stored in a database that is accessible from the application, or
(3) transmitted from systems external to the application in question

10.6.4 ANALYSIS OF THE WORK ENVIRONMENT

In some applications the user interface for a computer-based system is placed in a "user-friendly location" (e.g., proper lighting, good display height, easy keyboard access), but in others (e.g., a factory floor or an airplane cockpit), lighting may be suboptimal, noise may be a factor, a keyboard or mouse may not be an option, display placement may be less than ideal. The interface designer may be constrained by factors that mitigate against ease of use.

In addition to physical environmental factors, the workplace culture also comes into play. Will system interaction be measured in some manner (e.g., time per transaction or accuracy of a transaction)? Will two or more people have to share information before an input can be provided? How will support be provided to users of the system? These and many related questions should be answered before the interface design commences.

10.7 INTERFACE DESIGN STEPS

Once interface analysis has been completed, all tasks (or objects and actions) required by the end user have been identified in detail and the interface design activity commences. Interface design, like all software engineering design, is an iterative process.

The combination of the following steps are used for interface design models:

1. Using information developed during interface analysis, define interface objects and actions (operations).
2. Define events (user actions) that will cause the state of the user interface to change. Model this behavior.
3. Depict each interface state as it will actually look to the end user.
4. Indicate how the user interprets the state of the system from information provided through the interface.

10.7.1 APPLYING INTERFACE DESIGN STEPS

The definition of interface objects and the actions that are applied to them is an important step in interface design. To accomplish this, user scenarios are parsed. That is, a use case is written.

Once the objects and actions have been defined and elaborated iteratively, they are categorized by type. Target, source, and application objects are identified. A source object (e.g., a report icon) is dragged and dropped onto a target object (e.g., a printer icon). The implication of this action is to create a hard-copy report. An application object represents application-specific data that are not directly manipulated as part of screen interaction.

10.8 WEBAPP INTERFACE DESIGN

The user interface of a WebApp is its first impression. Regardless of the value of its content, the sophistication of its processing capabilities and services, and the overall benefit of the WebApp itself a poorly designed interface will disappoint the potential user. Because of the sheer volume of competing WebApps in virtually every subject area, the interface must grab a potential user immediately. Nielsen and Wagner suggest a few simple guide lines based on their redesign of a major WebApp:

- Server errors, even minor ones, are likely to cause a user to leave the Web site and look elsewhere for information or services.
- Reading speed on a computer monitor is approximately 25 percent slower than reading speed for hard copy. Therefore, do not force the user to read voluminous amounts of text, particularly when the text explains the operation of the WebApp or assists in navigation.
- Avoid under construction signs—they raise expectations and cause an unnecessary link that is sure to disappoint.

- Users prefer not to scroll. Important information should be placed within the dimensions of a typical browser window.
- Navigation menus and head bars should be designed consistently and should be available on all pages that are available to the user. The design should not rely on browser functions to assist in navigation.
- Aesthetics should never supersede functionality. For example, a simple button might be a better navigation option than an aesthetically pleasing, but vague image or icon whose intent is unclear.
- Navigation options should be obvious, even to the casual user. The user should not have to search the screen to determine how to link to other content or services.

A well-designed interface improves the user's perception of the content or services provided by the site. It need not be flashy, but it should always be well structured and ergonomically sound.

10.9 DESIGN EVALUATION

Once an operational user interface prototype has been created, it must be evaluated to determine whether it meets the needs of the user. Evaluation can span a formality spectrum that ranges from an informal test drive, in which a user provides vamped feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end-users.

The user interface evaluation cycle takes the form shown in Figure 10.3. After the design model has been completed, a first-level prototype is created. The prototype is evaluated by the user, who provides the designer with direct comments about the efficacy of the interface. In addition, if formal evaluation techniques are used like questionnaires, rating sheets, the designer may extract information from these data.

Design modifications are made based on user input and the next level prototype is created. The evaluation cycle continues until no further modifications to the interface design are necessary.

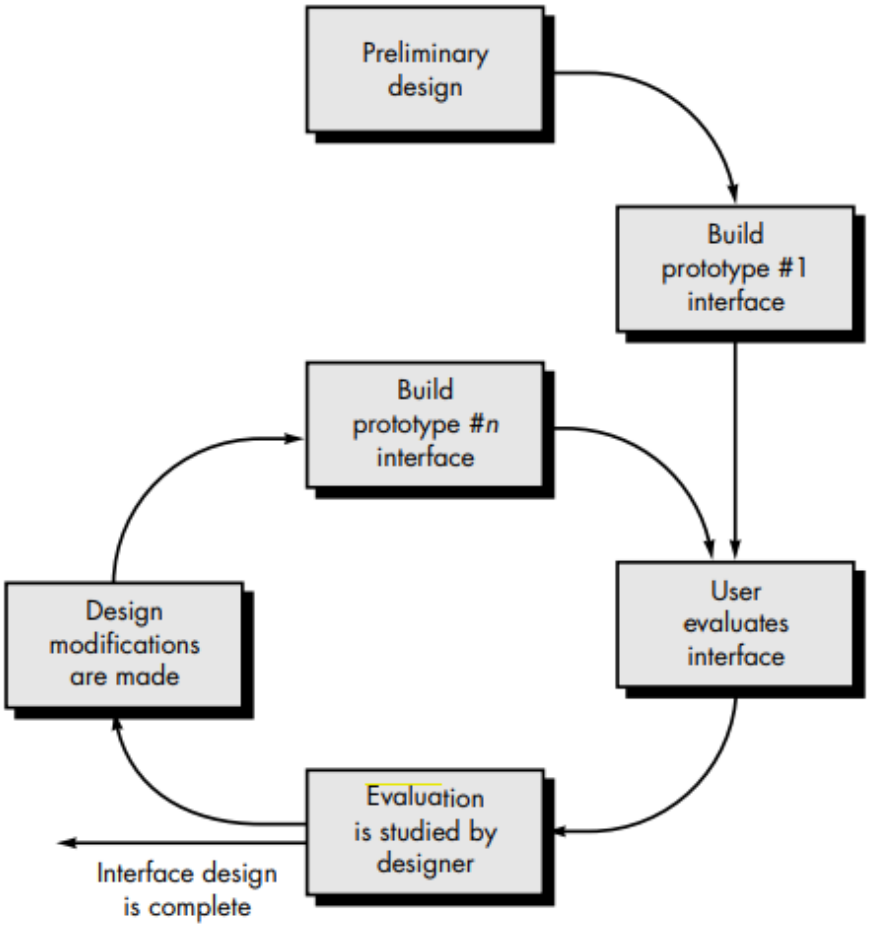


Fig 10.3 The interface design evaluation cycle

The prototyping approach is effective, if potential problems can be uncovered and corrected early, the number of loops through the evaluation cycle will be reduced and development time will shorten. If a design model of the interface has been created, a number of evaluation criteria can be applied during early design reviews:

1. The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.
4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

Once the first prototype is built, the designer can collect a variety of qualitative and quantitative data that will assist in evaluating the interface. To collect qualitative data, questionnaires can be distributed to users of the prototype. Questions can be all (1) simple yes/no response, (2) numeric response, (3) Scale or subjective response, or (4) percentage or subjective response.

Examples are:

1. Were the icons self-explanatory? If not, which icons were unclear?
2. Were the actions easy to remember and to invoke?
3. How many different actions did you use?
4. How easy was it to learn basic system operations (scale 1 to 5)?
5. Compared to other interfaces you've used, how would this rate—top 1%, top 10%, top 25%, top 50%, bottom 50%?

If quantitative data are desired, a form of time study analysis can be conducted. Users are observed during interaction, and data—such as number of tasks correctly completed over a standard time period, frequency of actions, sequence of actions, time spent looking at the display, number and types of errors, error recovery time, time spent using help, and number of help references per standard time period—are collected and used as a guide for interface modification.

10.10 CHECK YOUR PROGRESS

1. Categories of user interface design are _____ and _____.
2. The prototype is evaluated by _____.
3. Design evaluation is studied by _____
4. After the first prototype was built, the designer collects _____ and _____ data that will assist to evaluate interface.
5. The qualitative data _____ can be distributed to the user.
6. _____ is the front-end application view to which user interacts in order to use the software.

Answers to check your progress:

1. Command Line Interface and Graphical User Interface
2. User
3. Designer

4. Qualitative and quantitative
5. Questionnaires
6. Use interface design

10.11 SUMMARY

The user interface is arguably the most important element of a computer-based system or product. If the interface is poorly designed, the user's ability to tap the computational power of an application may be severely hindered. In fact, a weak interface may cause to fail the well designed and implemented application.

Three important principles guide the design of effective user interfaces:
(1) Place the user in control,
(2) Reduce the user's memory load, and
(3) Make the interface consistent.
To achieve an interface that abides by these principles, an organized design process must be conducted. User interface design begins with the identification of user, task, and environmental requirements. Task analysis is a design activity that defines user tasks and actions using either an elaborative or object-oriented approach.

Once tasks have been identified, user scenarios are created and analyzed to define a set of interface objects and actions. This provides a basis for the creation of screen layout that depicts graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. A variety of implementation tools are used to build a prototype for evaluation by the user.

The user interface is the window into the software. In many cases, the interface molds a user's perception of the quality of the system. If the window is smudged, wavy, or broken, the user may reject an otherwise powerful computer-based system.

10.12 KEYWORDS

1. **Use interface (UI) design-** is the front-end application view to which user interacts in order to use the software.
2. **Command Line Interface-** an interface that provides a command prompt, where the

user types the command and feeds to the system.

3. **Graphical User Interface-** an interface that provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software.
4. **Mnemonics-** means the keyboard shortcuts to do some action on the screen.

10.13 QUESTIONS FOR SELF STUDY

1. What is user interface?
2. Discuss User interface design activities.
3. Mention the categories of user interface.
4. Discuss the four activities of frame work of UI design using spiral model.
5. Discuss golden rules to design interface.
6. Explain in detail webapp user interface design.
7. Elaborate on design evaluation.
8. Elucidate qualitative and quantitative data collection process in design evaluation.
9. Discuss important principles to guide design effective user interface.
10. Write a short note on interface analysis.

10.14 REFERENCES

1. Software Engineering, A Practitioner's Approach – 7th Edition, Roger S.Pressman
2. Software Project Management in Practice – Pankaj Jalote
3. Software Engineering : Pearson New International Edition – Ian Sommerville, 2013.

UNIT-11: PROJEC MANAGEMNT CONCEPTS

STRUCTURE

11.0 OBJECTIVES

After studying this unit, you will be able to:

- Explain project management concepts
- Learns the important attributes of project management
- Appraise the project management spectrum.
- Discuss software project planning
- Study the different roles of software engineers in PM.
- Learns to handle the risks and risk avoidance methods

11.1 INTRODUCTION

In this unit, we are going to discuss about project management concepts. Software project management is an umbrella activity within software engineering. It begins before any technical activity is initiated and continues throughout the definition, development, and support of computer software Four P's have a substantial influence on software project management—people, product, process, and project.

11.2 INTRODUCTION TO PROJECT MANAGEMENT

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled. It is a procedure of managing, allocating and timing resources to develop computer software that fulfills requirements. In software Project Management, the client and the developers need to know the length, period and cost of the project.

Prerequisite of software project management

Below figure 11.1 shows the three needs for software project management are

Figure 11.1 pre requisites for software project management

- Time
- Cost
- Quality

It is an essential part of the software organization to deliver a quality product, keeping the cost within the client's budget and deliver the project as per schedule. There are various factors, both external and internal, which may impact this triple factor. Any of three-factor can severely affects the other two.

11.3 THE MANAGEMENT SPECTRUM

For properly building a product, there's a very important concept that we all should know in software project planning while developing a product. The four critical components are shown in figure 11.2 These 4 critical components in software project planning are known as the 4P's namely:

- Product
- Process
- People
- Project

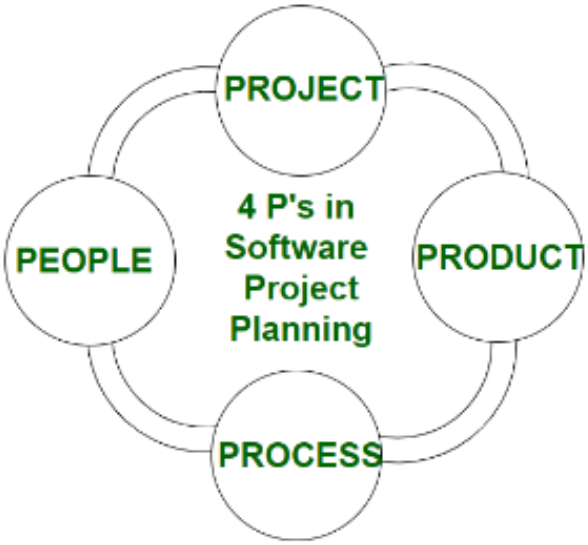


Figure 11.2 Four P's of Software project planning

These components play a very important role in your project that can help the team to meet its goals and objective. Now, let's dive into each of them a little in detail to get a better understanding:

People

The most important component of a product and its successful implementation is human resources. In building a proper product, a well-managed team with clear-cut roles defined for each person or team will lead to the success of the product. We need to have a good team in order to save our time, cost, and effort. Some assigned roles in software project planning are project manager, team leaders, stakeholders, analysts, and other IT professionals. Managing people successfully is a tricky process which a good project manager can do.

Product

As the name inferred, this is the deliverable or the result of the project. The project manager should clearly define the product scope to ensure a successful result, control the team members, as well technical hurdles that he or she may encounter during the building of a product. The product can consist of either tangible or intangible such as shifting the company to a new place or getting new software in a company.

Process

In every planning, a clearly defined process is the key to the success of any product. It regulates how the team will go about its development in the respective time period. The Process has several steps involved like, documentation phase, implementation phase, deployment phase, and interaction phase.

Project

The last and final P in software project planning is Project. In this phase, the project manager plays a critical role. They are responsible to guide the team members to achieve the project's target and objectives, helping & assisting them with issues, checking on cost and budget, and making sure that the project stays on track with the given deadlines.

11.4 PEOPLE

The cultivation of motivated, highly skilled software people has been discussed since the 1960s. The people factor is so important that the Software Engineering Institute has developed a people management capability maturity model (PM-CMM), to enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability.

The people management maturity model defines the following key practice areas for software people: recruiting, selection, performance management, training, compensation, career development, organization and work design, and team/culture development. Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

Project Manager

A project manager is a character who has the overall responsibility for the planning, design, execution, monitoring, controlling and closure of a project. A project manager represents an essential role in the achievement of the projects.

A project manager is a character who is responsible for giving decisions, both large and small projects. The project manager is used to manage the risk and minimize uncertainty. Every decision the project manager makes must directly profit their project.

Role of a Project Manager:

1. Leader

A project manager must lead his team and should provide them direction to make them understand what is expected from all of them.

2. Medium:

The Project manager is a medium between his clients and his team. He must coordinate and transfer all the appropriate information from the clients to his team and report to the senior management.

3. Mentor:

He should be there to guide his team at each step and make sure that the team has an attachment. He provides a recommendation to his team and points them in the right direction.

Responsibilities of a Project Manager:

1. Managing risks and issues.
2. Create the project team and assigns tasks to several team members.
3. Activity planning and sequencing.
4. Monitoring and reporting progress.
5. Modifies the project plan to deal with the situation.

11.4.1 THE PLAYERS

The software process or software project is populated by players, who can be categorized into one of five constituencies,

1. Senior managers who define the business issues that often have significant influence on the project.
2. Project (technical) managers who must plan, motivate, organize, and control the practitioners who do software work.
3. Practitioners who deliver the technical skills that are necessary to engineer a product or application.
4. Customers who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.

5. End-users who interact with the software once it is released for production use.

To be effective, the project team must be organized in a way that maximizes each person's skills and abilities. And that's the job of the team leader.

11.4.2 TEAM LEADERS

Jerry Weinberg suggests a MOI model of leadership:

- **Motivation:** The ability to encourage technical people to produce to their best ability.
- **Organization:** The ability to mold existing processes or invent new ones that will enable the initial concept to be translated into a final product.
- **Ideas or innovation:** The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

A software project manager should concentrate on understanding the problem to be solved, managing the flow of ideas, and at the same time, letting everyone on the team know that quality counts and that it will not be compromised. Four key essential qualities of a project manager are:

- **Problem solving:** An effective software project manager can diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution, apply lessons learned from past projects to new situations, and remain flexible enough to change direction if initial attempts at problem solution are fruitless.
- **Managerial identity:** A good project manager must take charge of the project. He or She must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.
- **Achievement:** To optimize the productivity of a project team, a manager must reward initiative and accomplishment and demonstrate through his own actions that controlled risk taking will not be punished.
- **Influence and team building:** An effective project manager must be able to read people; project manager must be able to understand verbal and nonverbal signals and react to the needs of the people sending these signals. The manager must remain under control in high-stress situations.

11.4.3 THE SOFTWARE TEAM

There are almost as many human organizational structures for software development as there are organizations that develop software. However, the organization of the people directly involved in a new software project is within the project manager's purview. The following options are available for applying human resources to a project that will require n people working for k years:

1. n individuals are assigned to m different functional tasks, relatively little combined work occurs; coordination is the responsibility of a software manager who may have six other projects to be concerned with.
2. n individuals are assigned to m different functional tasks (m < n) so that informal teams are established; an ad hoc team leader may be appointed; coordination among teams is the responsibility of a software manager.
3. n individuals are organized into t teams; each team is assigned one or more functional tasks; each team has a specific structure that is defined for all teams working on a project; coordination is controlled by both the team and a software project manager.

The best team structure depends on the management style of your organization, the number of people who will populate the team and their skill levels, and the overall problem difficulty. Mantei suggests three generic team organizations.

- **Democratic decentralized (DD):** This software engineering team has no permanent leader. Rather, task coordinators are appointed for short durations and then replaced by others who may coordinate different tasks. Decisions on problems and approach are made by group consensus. Communication among team members is horizontal.
- **Controlled decentralized (CD):** This software engineering team has a defined leader who coordinates specific tasks and secondary leaders that have responsibility for subtasks. Problem solving remains a group activity, but implementation of solutions is partitioned among subgroups by the team leader. Communication among subgroups and individuals is horizontal. Vertical communication along the control hierarchy also occurs.
- **Controlled Centralized (CC):** Top-level problem solving and internal team coordination are managed by a team leader. Communication between the leader and team members is vertical.

Mantei describes seven project factors that should be considered when planning the structure of software engineering teams:

- The difficulty of the problem to be solved.
- The size of the resultant program(s) in lines of code or function points
- The time that the team will stay together (team lifetime).
- The degree to which the problem can be modularized.
- The required quality and reliability of the system to be built.
- The rigidity of the delivery date.
- The degree of sociability (communication) required for the project.

Because a centralized structure completes tasks faster, it is the most used at handling simple problems. Decentralized teams generate more and better solutions than individuals. Therefore such teams have a greater probability of success when working on difficult problems. Since the CD team is centralized for problem solving, either a CD or CC team structure can be successfully applied to simple problems. A DD structure is best for difficult problems.

A high-performance team must be a central goal of a software engineering organization. To achieve a high-performance team:

- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained.

The software project manager, working together with the team, should clearly refine roles and responsibilities before the project begins. The team itself should establish its own mechanisms for accountability and define a series of corrective approaches when a member of the team fails to perform.

Every software team experiences small failures. The key to avoiding an atmosphere of failure is to establish team-based techniques for feedback and problem solving. In addition, failure by any member of the team must be viewed as a failure by the team itself. This leads to a team-oriented approach to corrective action, rather than the finger-pointing and mistrust that grows rapidly on toxic teams.

11.4.4 COORDINATION AND COMMUNICATION ISSUES

There are many reasons that software projects get into trouble. The scale of many development efforts is large, leading to complexity, confusion, and significant difficulties in

coordinating team members. Uncertainty is common, resulting in a continuing stream of changes that ratchets the project team. Interoperability has become a key characteristic of many systems. New software must communicate with existing software and conform to predefined constraints imposed by the system or product.

These characteristics of modern software—scale, uncertainty, and interoperability—are facts of life. To deal with them effectively, a software engineering team must establish effective methods for coordinating the people who do the work. To accomplish this, mechanisms for formal and informal communication among team members and between multiple teams must be established. Formal communication is accomplished through writing, structured meetings, and other relatively no interactive and impersonal communication channels. Informal communication is more personal. Members of a software team share ideas on an ad hoc basis, ask for help as problems arise, and interact with one another on a daily basis.

Informal, interpersonal procedures include group meetings for information dissemination and problem solving and collocation of requirements and development staff.

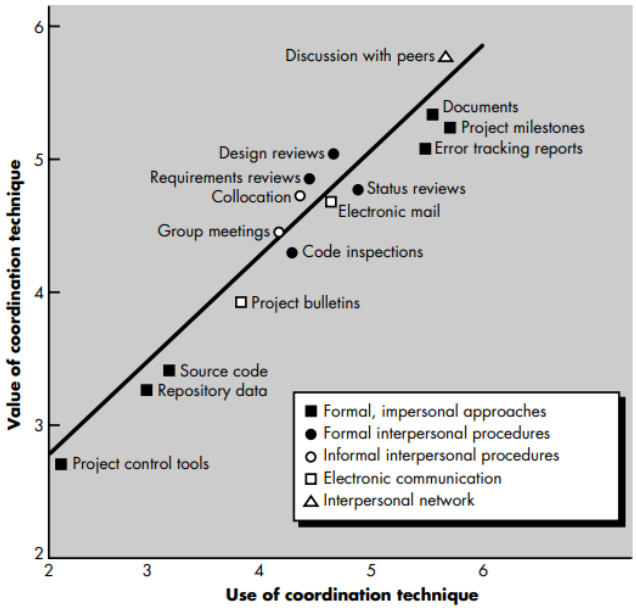


FIGURE 11.3 Values and Use of Coordination and Communication Techniques

Electronic communication encompasses electronic mail, electronic bulletin boards, and by extension, video-based conferencing systems.

Interpersonal networking includes informal discussions with team members and those outside the project who may have experience or insight that can assist team members.

To assess the efficacy of these techniques for project coordination, Kraul and Streeter studied 65 software projects involving hundreds of technical staff. Figure 11.3 expresses the value

and use of the coordination techniques just noted. Referring to figure, the perceived value (rated on a seven point scale) of various coordination and communication techniques is plotted against their frequency of use on a project. Techniques that fall above the regression line were judged to be relatively valuable, given the amount that they were used. Techniques that fell below the line were perceived to have less value. It is interesting to note that interpersonal networking was rated the technique with highest coordination and communication value. It is also important to note that early software quality assurance mechanisms (requirements and design reviews) were perceived to have more value than later evaluations of source code (code inspections).

11.5 THE PRODUCT

We must examine the product and the problem it is intended to solve at the very beginning of the project. At a minimum, the scope of the product must be established and bounded.

11.5.1 SOFTWARE SCOPE

The first software project management activity is the determination of software scope. Scope is defined by answering the following questions:

- **Context:** How does the software to be built fit into a larger system, product, or business context and what constraints are imposed as a result of the context.
- **Information objectives:** What customer-visible data objects are produced as output from the software? What data objects are required for input?
- **Function and performance:** What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Software project scope must be unambiguous and understandable at the management and technical levels. A statement of software scope must be bounded. That is, quantitative data like number of simultaneous users, size of mailing list, maximum allowable response time are stated explicitly; constraints and/or limitations such as product cost restricts memory size are noted, and mitigating factors are described.

11.5.2 PROBLEM DECOMPOSITION

Problem decomposition, sometimes called partitioning or problem elaboration, is an activity that sits at the core of software requirements analysis. During the scoping activity no attempt

is made to fully decompose the problem. Rather, decomposition is applied in two major areas: (1) the functionality that must be delivered and (2) the process that will be used to deliver it.

11.6 THE PROCESS

The generic phases that characterize the software process—definition, development, and support—are applicable to all software. The problem is to select the process model that is appropriate for the software to be engineered by a project team. Wide arrays of software engineering paradigms were discussed:

- The linear sequential model
- The prototyping model
- The RAD model
- The incremental model
- The spiral model
- The WINWIN spiral model
- The component-based development model
- The concurrent development model
- The formal methods model
- The fourth generation techniques model.

Melding the Product and the Process

Project planning begins with the melding of the product and the process. Each function to be engineered by the software team must pass through the set of framework activities that have been defined for a software organization. Assume that the organization has adopted the following set of framework activities:

- **Customer communication**—tasks required to establish effective requirements elicitation between developer and customer.
- **Planning**—tasks required to define resources, timelines, and other project related information.
- **Risk analysis**—tasks required to assess both technical and management risks.
- **Engineering**—tasks required to build one or more representations of the application.
- **Construction and release**—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

- **Customer evaluation**—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering activity and implemented during the construction activity.
- **Process Decomposition**

  A software team should have a significant degree of flexibility in choosing the software engineering paradigm that is best for the project and the software engineering tasks that populate the process model once it is chosen. A relatively small project that is similar to past efforts might be best accomplished using the linear sequential approach. If very tight time constraints are imposed and the problem can be heavily compartmentalized, the RAD model is probably the right option. If the deadline is so tight that full functionality cannot reasonably be delivered, an incremental strategy might be best. Similarly, projects with other characteristics (e.g., uncertain requirements, breakthrough technology, difficult customers, and significant reuse potential) will lead to the selection of other process models.

11.7 THE PROJECT

In order to manage a successful software project, we must understand what can go wrong so that problems can be avoided and how to do it right. John Reel defines ten signs that indicate that an information systems project is at risk:

1. Software people don't understand their customer's needs.
2. The product scope is poorly defined.
3. Changes are managed poorly.
4. The chosen technology changes.
5. Business needs change or is ill-defined.
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost or was never properly obtained.
9. The project team lacks people with appropriate skills.
10. Managers and practitioners avoid best practices and lessons learned.

Five-part commonsense approach to software projects:

1. *Start on the right foot:* This is accomplished by working hard to understand the problem that is to be solved and then setting realistic objects and expectations for everyone who will be involved in the project. It is reinforced by building the right team and giving the

team the autonomy, authority, and technology needed to do the job.

2. *Maintain momentum:* Many projects get off to a good start and then slowly disintegrate. To maintain momentum, the project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.
3. *Track progress:* For a software project, progress is tracked as work products (e.g., specifications, source code, sets of test cases) are produced and approved using formal technical reviews as part of a quality assurance activity.
4. *Make smart decisions:* In essence, the decisions of the project manager and the software team should be to "keep it simple." Whenever possible, decide to use commercial off-the-shelf software or existing software components, decide to avoid custom interfaces when standard approaches are available, decide to identify and then avoid obvious risks, and decide to allocate more time than you think is needed to complex or risky tasks.
5. *Conduct a postmortem analysis:* Establish a consistent mechanism for extracting lessons learned for each project. Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from team members and customers, and record findings in written form.

11.8 W5HH PRINCIPLE

We need an organizing principle that scales down to provide simple project plans for simple projects. Boehm suggests an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources. He calls it the WWWWWHH principle, after a series of questions that lead to a definition of key project characteristics and the resultant project plan:

- **Why is the system being developed?** The answer to this question enables all parties to assess the validity of business reasons for the software work. Stated in another way, does the business purpose justify the expenditure of people, time, and money?
- **What will be done, by when?** The answers to these questions help the team to establish a project schedule by identifying key project tasks and the milestones that are required by the customer.
- **Who is responsible for a function?** Earlier in this chapter, we noted that the role and

responsibility of each member of the software team must be defined. The answer to this question helps accomplish this.

- **Where they are organizationally located?** Not all roles and responsibilities reside within the software team itself. The customer, users, and other stakeholders also have responsibilities. **How will the job be done technically and managerially?** Once product scope is established, a management and technical strategy for the project must be defined.
- **How much of each resource is needed?** The answer to this question is derived by developing estimates based on answers to earlier questions.

Boehm's W5HH principle is applicable regardless of the size or complexity of a software project. The questions noted provide an excellent planning outline for the project manager and the software team.

11.9 CRITICAL PRACTICES

The Airlie Council8 has developed a list of critical software practices for performance-based management. These practices are consistently used by, and considered critical by, highly successful software projects and organizations whose bottom line performance is consistently much better than industry averages. In an effort to enable a software organization to determine whether a specific project has implemented critical practices, the Airlie Council has developed a set of Quick Look questions for a project:

- *Formal risk management:* What are the top ten risks for this project? For each of the risks, what is the chance that the risk will become a problem and what is the impact if it does?
- *Empirical cost and schedule estimation:* What is the current estimated size of the application software (excluding system software) that will be delivered into operation? How was it derived?
- *Metric-based project management:* Do you have in place a metrics program to give an early indication of evolving problems? If so, what is the current requirements volatility?
- *Earned value tracking:* Do you report monthly earned value metrics? If so, are these metrics computed from an activity network of tasks for the entire effort to the next delivery?

- *Defect tracking against quality targets:* Do you track and periodically report the number of defects found by each inspection (formal technical review) and execution test from program inception and the number of defects currently closed and open?
- *People-aware program management:* What is the average staff turnover for the past three months for each of the suppliers/developers involved in the development of software for this system?

If a software project team cannot answer these questions or answers them inadequately, a thorough review of project practices is indicated.

11.10 CHECK YOUR PROGRESS

1. What are three needs for Software project management?
2. What are the 4 P's of software project planning?
3. _____ represents an essential role in the achievements of the projects.
4. Project planning begins with the melding of the _____and the_____
5. What are the generic phases that characterize the software process?
6. What are the two major areas where problem decomposition can be applied?

Answers to Check Your Progress

1. Cost, time and quality
2. Process, people. Product and project.
3. A project manager
4. Product and process
5. Definition, development and support.
6. i. the functionality that must be delivered and
   ii. the process will be used to delivered it

11.11 SUMMARY

Software project management is an umbrella activity within software engineering. It begins before any technical activity is initiated and continues throughout the definition, development, and support of computer software Four P's have a substantial influence on software project management—people, product, process, and project. People must be organized into effective teams, motivated to do high-quality software work, and coordinated to achieve effective communication. The product requirements must be communicated from

customer to developer, partitioned (decomposed) into their constituent parts, and positioned for work by the software team. The process must be adapted to the people and the problem. A common process framework is selected, an appropriate software engineering paradigm is applied, and a set of work tasks is chosen to get the job done. Finally, the project must be organized in a manner that enables the software team to succeed.

The pivotal element in all software projects is people. Software engineers can be organized in a number of different team structures that range from traditional control hierarchies to open paradigm teams. A variety of coordination and communication techniques can be applied to support the work of the team. In general, formal reviews and informal person-to-person communication have the most value for practitioners. The project management activity encompasses measurement and metrics, estimation, risk analysis, schedules, tracking, and control.

11.12 KEYWORDS

- **People:** Human resources are the important component of successful implementation
- **Product:** Deliverable project
- **Leader:** A project manager who leads the team
- **Mentor:** The one who guides the team.

11.13 QUESTIONS FOR SELF STUDY

1. What are the pre requisites for software project management?
2. What are the critical components in software project planning?
3. Discuss four P's of software planning.
4. Discuss the role of project manager.
5. What are the three generic phase's team organizations?
6. How can you define software scope in software project management activity.
7. Discuss the responsibilities of project manager.
8. Explain problem decomposition.
9. What are the set of framework activities have to be defined by software team for a software organization?
10. Explain process decomposition.
11. Discuss risk indication signs.
12. What are the five commonsense approaches to software projects?

13. Discuss W5HH principal.
14. Discuss critical practices.

11.14 REFERENCES

1. Software Engineering, A Practitioner's Approach – 7th Edition, Roger S.Pressman
2. Software Project Management in Practice – Pankaj Jalote
3. Software Engineering : Pearson New International Edition – Ian Sommerville, 2013

UNIT-12: ESTIMATION FOR SOFTWARE PROJECT MANAGEMENT

STRUCTURE

12.0 OBJECTIVES

After studying this unit, you will be able to:

- Discuss about how the software project planner must estimate cost, time, people, resources and risk involved.
- Describe objectives of software project planning.
- Use Decomposition Techniques

- Explain the scope of software, feasibility, availability of resources.
- Discuss techniques involved in software project estimation.

12.1 INTRODUCTION

In this unit, we are going to discuss about estimation for software project management. The software project planner must estimate three things before a project begins: how long it will take, how much effort will be required, and how many people will be involved. In addition, the planner must predict the resources (hardware and software) that will be required and the risk involved.

12.2 INTRODUCTION TO ESTIMATION

Software project management begins with a set of activities that are collectively called project planning. Before the project can begin, the manager and the software team must estimate

- The work to be done,
- The resources that will be required,
- The time that will elapse from start to finish.

Whenever estimates are made, we look into the future and accept some degree of uncertainty as a matter of course.

Although estimating is as much art as it is science, this important activity need not be conducted in a haphazard manner. Useful techniques for time and effort estimation do exist. Process and project metrics can provide historical perspective and powerful input for the generation of quantitative estimates. Past experience (of all people involved) can aid immeasurably as estimates are developed and reviewed. Because estimation lays a foundation for all other project planning activities and project planning provides the road map for successful software engineering.

Project planning involves estimation of your attempt to determine

- how much money
- how much effort
- how many resources and
- how much time

It will take to build a specific software-based system or product. Software managers will do this using

- information solicited from customers and software engineers and
- Software metrics data collected from past projects.

It would seem reasonable to develop an estimate before you start creating the software because most computer-based systems and products cost considerably more to build.

Estimation-

- Begins with a description of the scope of the product. Until the scope is "bounded" it's not possible to develop a meaningful estimate.
- The problem is then decomposed into a set of smaller problems and each of these is estimated using historical data and experience as guides. It is advisable to generate your estimates using at least two different methods (as a cross check).
- Problem complexity and risk are considered before a final estimate is made.
- A simple table delineating the tasks to be performed, the functions to implemented, and the cost, effort, and time involved for each is generated.
- A list of required project resources is also produced.

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

The four basic steps in Software Project Estimation are −

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

12.3 OBSERVATION ON ESTIMATION

Estimation of resources, cost, and schedule for a software engineering effort requires

- Experience,
- Access to good historical information,
- The courage to commit to quantitative predictions when qualitative information is all that exists.
- Identify inherent risk that may lead to uncertainty.

Factors that affect the estimation process:

(a) Project complexity-

- Has a strong effect on the uncertainty inherent in planning.
- Is a relative measure that is affected by familiarity with past effort?
- More subjective assessments of complexity can be established early in the planning process.

(b) Project size-

- is another important factor that can affect the accuracy and efficacy of estimates.
- As size increases, the interdependency among various elements of the software grows rapidly.
- Problem decomposition, an important approach to estimating, becomes more difficult because decomposed elements may still be formidable.
- The degree of structural uncertainty also has an effect on estimation risk.

(c) The availability of historical information

- Has a strong influence on estimation risk.
- By looking back, we can emulate things that worked and improve areas where problems arose.
- When comprehensive software metrics are available for past projects, estimates can be made with greater assurance, schedules can be established to avoid past difficulties, and overall risk is reduced.

Risk is measured by the degree of uncertainty in the quantitative estimates established for resources, cost, and schedule. If project scope is poorly understood or project requirements are subject to change, uncertainty and risk become dangerously high.

The software planner should demand-

- completeness of function
- performance and
- Interface definitions (contained in a System Specification).

The planner, and more important, the customer should recognize that variability in software requirements means instability in cost and schedule.

12.4 THE PROJECT PLANNING PROCESS

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of:

- resources
- cost
- Schedule.

These estimates are made within a limited time frame at the beginning of a software project and should be updated regularly as the project progresses. In addition, estimates should attempt to define best case and worst case scenarios so that project outcomes can be bounded. The planning objective is achieved through a process of information discovery that leads to reasonable estimates.

12.5 SOFTWARE SCOPE AND FEASIBILITY

The first activity in software project planning is the determination of software scope. Function and performance allocated to software during system engineering should be assessed to establish a project scope that is unambiguous and understandable at the management and technical levels. A statement of software scope must be bounded. Software scope describes

- the data and control to be processed,
- function,
- performance,
- constraints,
- interfaces and
- Reliability.

Functions described in the statement of scope are evaluated and in some cases refined to provide more detail prior to the beginning of estimation. Because both cost and schedule estimates are functionally oriented, some degree of decomposition is often useful.

Performance considerations encompass

- processing and
- Response time requirements.

Constraints identify limits placed on the

- software by external hardware,
- available memory, or
- Other existing systems.

12.5.1 OBTAINING INFORMATION NECESSARY FOR SCOPE

The most commonly used technique to bridge the communication gap between the customer and developer and to get the communication process started is to

(a) conduct a preliminary meeting or interview:

The first meeting between the software engineer (the analyst) and the customer can be likened to the awkwardness of a first date between two adolescents. Neither person knows what to say or ask; both are worried that what they do say will be misinterpreted; both are thinking about where it might lead (both likely have radically different expectations here); both want to get the thing over with; but at the same time, both want it to be a success.

Yet, communication must be initiated. Gause and Weinberg suggest that the analyst start by asking context-free questions; that is, a set of questions that will lead to a basic understanding of the problem, the people who want a solution, the nature of the solution desired, and the effectiveness of the first encounter itself.

The first set of context-free questions focuses on the customer, the overall goals and benefits. For example, the analyst might ask:

- Who is behind the request for this work?
- Who will use the solution?
- What will be the economic benefit of a successful solution?
- Is there another source for the solution?

The next set of questions enables the analyst to gain a better understanding of the problem and the customer to voice any perceptions about a solution:

- How would you (the customer) characterize "good" output that would be generated by a successful solution?
- What problem(s) will this solution address?
- Can you show me (or describe) the environment in which the solution will be used?

The final set of questions focuses on the effectiveness of the meeting. Gause and Weinberg call these "meta-questions" and propose the following (abbreviated) list:

- Are you the right person to answer these questions? Are answers "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Can anyone else provide additional information?

The question and answer meeting format is not an approach that has been overwhelmingly successful. In fact, the Q&A session should be used for the first encounter only and then be replaced by a meeting format that combines elements of problem solving, negotiation, and specification.

(b) Communication through meeting format:

Customers and software engineers often have an unconscious "us and them" mindset. Rather than working as a team to identify and refine requirements, each constituency defines its own "territory" and communicates through a series of memos, formal position papers, documents, and question and answer sessions. History has shown that this approach works poorly. Misunderstandings abound, important information is omitted, and a successful working relationship is never established.

12.5.2 FEASIBILITY

Software feasibility has four solid dimensions:

Technology—

- Is a project technically feasible?
- Is it within the state of the art?
- Can defects be reduced to a level matching the application's needs?

Finance—

- Is it financially feasible?
- Can development be completed at a cost the software organization, its client, or the market can afford?

Time—Will the project's time-to-market beat the competition?

Resources—Does the organization have the resources needed to succeed?

For some projects in established areas the answers are easy.

- You have done projects like this one before. After a few hours or sometimes a few weeks of investigation, you are sure you can do it again.

Projects on the margins of your experience are not so easy.

- A team may have to spend several months discovering what the central, difficult-to-implement requirements of a new application actually are.
- Do some of these requirements pose risks that would make the project infeasible? Can these risks be overcome?

- The feasibility team ought to carry initial architecture and design of the high-risk requirements to the point at which it can answer these questions.
- In some cases, when the team gets negative answers, a reduction in requirements may be negotiated.

12.6 RESOURCES

The second software planning task is estimation of the resources required to accomplish the software development effort. Fig 12.1 illustrates development resources as a pyramid.

- The development environment—hardware and software tools—sits at the foundation of the resources pyramid and provides the infrastructure to support the development effort.
- At a higher level, we encounter reusable software components—software building blocks that can dramatically reduce development costs and accelerate delivery.
- At the top of the pyramid is the primary resource—people. Each resource is specified with four characteristics:
  (a) description of the resource,
  (b) a statement of availability,
  (c) time when the resource will be required;
  (d) duration of time that resource will be applied.

The last two characteristics can be viewed as a time window. Availability of the resource for a specified window must be established at the earliest practical time.

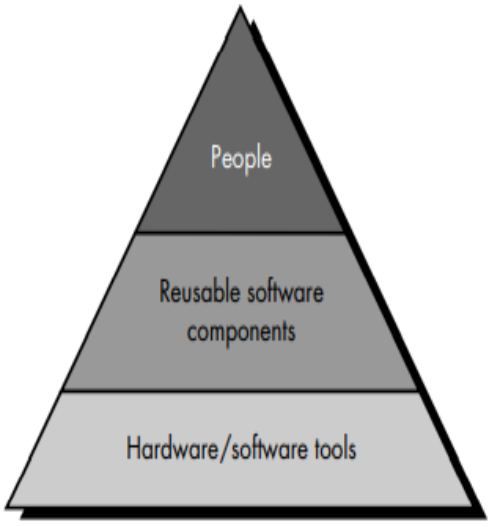


Fig 12.1. Development resources as a pyramid

12.6.1 CATEGORIES OF RESOURCES

Resources are majorly categorized into three, they are

(a) Human resources

- Planners need to select the number and the kind of people skills needed to complete the project. They need to specify the organizational position and job specialty for each person.
- Small projects of a few person-months may only need one individual.
- Large projects spanning many person-months or years require the location of the person to be specified also.
- The number of people required can be determined only after an estimate of the development effort.

(b) Development Environment Resources

- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products.
- Most software organizations have many projects that require access to the SEE provided by the organization.
- Planners must identify the time window required for hardware and software and verify that these resources will be available.

(c) Reusable Software Resources

- Component-based software engineering (CBSE) emphasizes reusability—that is, the creation and reuse of software building blocks.
- Such building blocks, often called components, must be cataloged for easy reference, standardized for easy application, and validated for easy integration.
- Bennatan suggests four software resource categories that should be considered as planning proceeds:

  (i) **Off-the-shelf components**: Existing software that can be acquired from a third party or that has been developed internally for a past project.

  (ii) **Full-experience components**: Existing specifications, designs, code, or test data developed for past projects that are similar to the software to be built for the current project. Members of the current software team have had full experience in the

application area represented by these components. Therefore, modifications required for full-experience components will be relatively low-risk.

(iii) **Partial-experience components**: Existing specifications, designs, code, or test data developed for past projects that are related to the software to be built for the current project but will require substantial modification. Members of the current software team have only limited experience in the application area represented by these components. Therefore, modifications required for partial-experience components have a fair degree of risk.

(iv) **New components**: Software components that must be built by the software team specifically for the needs of the current project.

The following guidelines should be considered by the software planner when reusable components are specified as a resource:

- If **off-the-shelf components** meet project requirements, acquire them. The cost for acquisition and integration of off-the-shelf components will almost always be less than the cost to develop equivalent software. In addition, risk is relatively low.
- If **full-experience components** are available, the risks associated with modification and integration is generally acceptable.
- If **partial-experience components** are available, their use for the current project must be analyzed. If extensive modification is required before the components can be properly integrated with other elements of the software, proceed carefully—risk is high.

12.7 SOFTWARE PROJECT ESTIMATION

For complex, custom systems, a large cost estimation error can make the difference between profit and loss. Cost overrun can be disastrous for the developer.

Too many variables—human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it. However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk. To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete!)
2. Base estimates on similar projects that have already been completed.

3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

Decomposition techniques take a "divide and conquer" approach to software project estimation. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.

Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. A model is based on experience (historical data) and takes the form d = f (vi) where, d is one of a number of estimated values (e.g., effort, cost, project duration) and vi are selected independent parameters (e.g., estimated LOC or FP).

Automated estimation tools implement one or more decomposition techniques or empirical models. When combined with a graphical user interface, automated tools provide an attractive option for estimating.

12.8 DECOMPOSITION TECHNIQUES

Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece. For this reason, we decompose the problem, re-characterizing it as a set of smaller (and hopefully, more manageable) problem.

Estimation uses one or both forms of partitioning. But before an estimate can be made, the project planner must understand the scope of the software to be built and generate an estimate of its “size.”

12.8.1 SOFTWARE SIZING

The accuracy of a software project estimate is predicated on a number of things:

(1) The degree to which the planner has properly estimated the size of the product to be built;

(2) The ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects);

(3) The degree to which the project plan reflects the abilities of the software team; and

(4) The stability of product requirements and the environment that supports the software engineering effort.

The methods to achieve reliable size and cost estimates:

- LOC-based estimation
- FP-based estimation
- Empirical estimation models
- COCOMO

Function point sizing: The planner develops estimates of the information domain characteristics.

Standard component sizing: Software is composed of a number of different "standard components" that are generic to a particular application area.

For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions. The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component.

Change sizing. This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project. The planner estimates the number and type (e.g., reuse, adding code, changing code, deleting code) of modifications that must be accomplished. Using an effort ratio for each type of change, the size of the change may be estimated.

12.8.2 PROBLEM-BASED ESTIMATION

Lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation:

(1) as an estimation variable to "size" each element of the software and

(2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common.

The problems of lines of code (LOC)

– Different languages lead to different lengths of code

– It is not clear how to count lines of code

– A report, screen, or GUI generator can generate thousands of lines of code in minutes

– Depending on the application, the complexity of code is different

An Example of LOC-Based Estimation

As an example of LOC and FP problem-based estimation techniques, let us consider a software package to be developed for a computer-aided design application for mechanical components. A review of the System Specification indicates that the software is to execute on an engineering workstation and must interface with various computer graphics peripherals including a mouse, digitizer, high resolution color display and laser printer. Using the System Specification as a guide, a preliminary statement of software scope can be developed.

For our purposes, we assume that further refinement has occurred and that the following major software functions are identified:

- User interface and control facilities (UICF)
- Two-dimensional geometric analysis (2DGA)
- Three-dimensional geometric analysis (3DGA)
- Database management (DBM)
- Computer graphics display facilities (CGDF)
- Peripheral control function (PCF)
- Design analysis modules (DAM)

Following the decomposition technique for LOC, an estimation table, shown in Fig 12.2, is developed. A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the 3D geometric analysis function is optimistic—4600 LOC, most likely—6900 LOC, and pessimistic—8600 LOC.

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

Fig 12.2 Estimation table for the LOC method

Applying Equation,

$S = (s_{opt} + 4sm + s_{pess})/6$

the expected value for the 3D geometric analysis function is 6800 LOC. Other estimates are derived in a similar fashion. By summing vertically in the estimated LOC column, an estimate of 33,200 lines of code is established for the CAD system.

- A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC/pm.
- Based on a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the LOC estimate and the historical productivity data, the total estimated project cost is $431,000 and the estimated effort is 54 person-months 1.

An Example of FP-Based Estimation

Decomposition for FP-based estimation focuses on information domain values rather than software functions. Referring to the function point calculation table, the project planner estimates inputs, outputs, inquiries, files, and external interfaces for the CAD software. For the purposes of this estimate, the complexity weighting factor is assumed to be average. Fig 12.3 presents the results of this estimation.

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of inputs	20	24	30	24	4	97
Number of outputs	12	15	22	16	5	78
Number of inquiries	16	22	28	22	5	88
Number of files	4	4	5	4	10	42
Number of external interfaces	2	2	3	2	7	15
Count total						*320*

Fig 12.3 Estimating information domain values

Each of the complexity weighting factors is estimated and the complexity adjustment factor is computed.

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
On-line data entry	4
Input transaction over multiple screens	5
Master files updated on-line	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
Complexity adjustment factor	1.17

Finally, the estimated number of FP is derived:

$$FP_{estimated} = \text{count-total} \times [0.65 + 0.01 \times \Sigma (F_i)]$$
$$FP_{estimated} = 375$$

Figure 12.4 Derived estimated number of FP

The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of $8000 per month, the cost per FP is approximately $1230. Based on the LOC estimate and the historical productivity data, the total estimated project cost is $461,000 and the estimated effort is 58 person-months.

12.8.3 PROCESS-BASED ESTIMATION

In process based estimation, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.

Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of software process activities must be performed for each function. Functions and related software process activities may be represented as part of a table similar to the one presented in Fig 12.4.

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table in Fig 12.5 average labor rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labor rate will vary for each task. Senior staffs heavily involved in early activities are generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
Task →				Analysis	Design	Code	Test		
Function									
↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

Fig 12.5 Process-based estimation table

Costs and effort for each function and software process activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, we now have two or three estimates for cost and effort that may be compared and reconciled.

12.9 EMPHERICAL ESTIMATION MODELS

An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP.

The empirical data that support most estimation models are derived from a limited sample of projects. For this reason, no estimation model is appropriate for all classes of software and in all development environments.

The Structure of Estimation Models

A typical estimation model is derived using regression analysis on data collected from past software projects. The overall structure of such models takes the form

$E = A + B \times (ev)C$

where A, B, and C are empirically derived constants, E is effort in person-months, and ev is the estimation variable (either LOC or FP). In addition to the relationship noted in Equation, the majority of estimation models have some form of project adjustment component that enables E to be adjusted by other project characteristics (e.g., problem complexity, staff

experience, development environment). Among the many LOC-oriented estimation models proposed in the literature are

E = 5.2 x (KLOC)0.91 Walston-Felix model

E = 5.5 + 0.73 x (KLOC)1.16 Bailey-Basili model

E = 3.2 x (KLOC)1.05 Boehm simple model

E = 5.288 x (KLOC)1.047 Doty model for KLOC > 9

FP-oriented models have also been proposed. These include

E = ————————————13.39 + 0.0545 FP Albrecht and Gaffney model

E = 60.62 x 7.728 x 10-8 FP3 Kemerer model

E = 585.7 + 15.12 FP Matson, Barnett, and Mellichamp model

A quick examination of these models indicates that each will yield a different result for the same values of LOC or FP.

THE COCOMO MODEL

Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for COnstructive COst MOdel. The original COCOMO model became one of the most widely used and discussed software cost estimation models in the industry. It has evolved into a more comprehensive estimation model, called COCOMO II. Like its predecessor, COCOMO II is actually a hierarchy of estimation models that address the following areas:

Application composition model: Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.

Early design stage model: Used once requirements have been stabilized and basic software architecture has been established.

Post-architecture-stage model: Used during the construction of the software. Like all estimation models for software, the COCOMO II models require sizing information. Three different sizing options are available as part of the model hierarchy: (a) object points, (b) function points and (c) lines of source code.

The COCOMO II application composition model uses object points and is illustrated in the following paragraphs. It should be noted that other, more

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

Figure 12.6 Table of complexity weighting for object types

Sophisticated estimation models (using FP and KLOC) are also available as part of COCOMO II. Like, function points, the object point is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports (3) components. Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e., simple, medium, or difficult) using criteria suggested by Boehm. In essence, complexity is a function of the number and source of the client and server data tables that are required to generate the screen or report and the number of views or sections presented as part of the screen or report. Once complexity is determined, the number of screens, reports, and components are weighted according to table in the figure12.7. The object point count is then determined by multiplying the original number of object instances by the weighting factor in table in figure 12.7 and summing to obtain a total object point count. When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted using the following formula: NOP = (object points) x [(100-%reuse)/100], where, NOP is defined as new object points.

To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived. Table 12.8 presents the productivity rate for different levels of developer experience and development environment maturity. Once the productivity rate has been determined, an estimate of project effort can be derived as **estimated effort = NOP/PROD**

PROD = NOP/person-month

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

Figure 12.8 Table for productivity rates for object points

The Software Equation

The software equation is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project. The model has been derived from productivity data collected for over 4000 contemporary software projects. Based on these data, an estimation model of the form

E = [LOC × B0.333/P]3 × (1/t4) (5-3) where E = effort in person-months or person-years
t = project duration in months or years
B = "special skills factor"16
P = "productivity parameter" that reflects:

- Overall process maturity and management practices
- The extent to which good software engineering practices are used
- The level of programming languages used
- The state of the software environment
- The skills and experience of the software team
- The complexity of the application

Typical values might be P = 2,000 for development of real-time embedded software; P = 10,000 for telecommunication and systems software; P = 28,000 for business systems applications. The productivity parameter can be derived for local conditions using historical data collected from past development efforts.

It is important to note that the software equation has two independent parameters:

(1) an estimate of size (in LOC) and
(2) an indication of project duration in calendar months or years.

To simplify the estimation process and use a more common form for their estimation model, Putnam and Myers suggest a set of equations derived from the software equation. Minimum development time is defined as

tmin = 8.14 (LOC/P)0.43 in months for tmin > 6 months
where, t is represented in years, P=12,000
E = 180 Bt3 in person-months for E ≥ 20 person-months
tmin = 8.14 (33200/12000)0.43
tmin = 12.6 calendar months

E = 180 × 0.28 × (1.05)3

E = 58 person-months

The results of the software equation correspond favorably with the estimates developed.

12.10 ESTIMATION FOR OBJECT ORIENTED PROJECTS

It is worthwhile to supplement conventional software cost estimation methods with a technique (as shown the table in figure 12.9) that has been designed explicitly for OO software. Lorenz and Kidd suggest the following approach:

1. Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
2. Using the requirements model, develop use cases and determine a count. Recognize that the number of use cases may change as the project progresses.
3. From the requirements model, determine the number of key classes.
4. Categorize the type of interface for the application and develop a multiplier for support classes:

Interface Type	Multiplier
No GUI	2.0
Text-based user interface	2.25
GUI	2.5
Complex GUI	3.0

Figure 12.9 Estimation table

   Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
5. Multiply the total number of classes (key + support) by the average number of work units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
6. Cross-check the class-based estimate by multiplying the average number of work units per use case.

12.11 SPECIALIZED ESTIMATION TECHNIQUES

When a software team encounters an extremely short project duration (weeks rather than months) that is likely to have a continuing stream of changes, project planning in general and estimation in particular should be abbreviated.

12.11.1 ESTIMATION FOR AGILE DEVELOPMENT

Because the requirements for an agile project are defined by a set of user scenarios (e.g., stories in Extreme Programming), it is possible to develop an estimation approach that is informal, reasonably disciplined, and meaningful within the context of project planning for each software increment. Estimation for agile projects uses a decomposition approach that encompasses the following steps:

1. Each user scenario (the equivalent of a mini use case created at the very start of a project by end users or other stakeholders) is considered separately for estimation purposes.
2. The scenario is decomposed into the set of software engineering tasks that will be required to develop it.

3a. the effort required for each task is estimated separately. Note: Estimation can be based on historical data, an empirical model, or experience.

3b. the volume of the scenario can be estimated in LOC, FP, or some other volume-oriented measure (e.g., use-case count).

4a. Estimates for each task are summed to create an estimate for the scenario.

4b. alternatively, the volume estimate for the scenario is translated into effort using historical data.

5. The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

Because the project duration required for the development of a software increment is quite short (typically three to six weeks), this estimation approach serves two purposes:

(1) to be certain that the number of scenarios to be included in the increment conforms to the available resources, and

(2) to establish a basis for allocating effort as the increment is developed.

12.11.2 ESTIMATION FOR WEBAPP PROJECTS

WebApp projects often adopt the agile process model. A modified function point measure, coupled with the steps outlined in the previous section, can be used to develop an estimate for the WebApp. Roetzheim suggests the following approach when adapting function points for WebApp estimation:

- Inputs are each input screen or form (for example, CGI or Java), each maintenance screen, and if you use a tab notebook metaphor anywhere, each tab.

- Outputs are each static Web page, each dynamic Web page script (for example, ASP,
- ISAPI, or other DHTML script), and each report (whether Web based or administrative in nature).
- Tables are each logical table in the database plus, if you are using XML to store data in a file, each XML object (or collection of XML attributes).
- Interfaces retain their definition as logical files (for example, unique record formats) into our out-of-the-system boundaries.
- Queries are each externally published or use a message-oriented interface. A typical example is DCOM or COM external references.

Function points (interpreted in the manner noted) are a reasonable indicator of volume for a WebApp.

Mendes and her colleagues suggest that the volume of a WebApp is best determined by collecting measures (called "predictor variables") associated with the application (e.g., page count, media count, function count), its Web page characteristics (e.g., page complexity, linking complexity, graphic complexity), media characteristics (e.g., media duration), and functional characteristics (e.g., code length, reused code length). These measures can be used to develop empirical estimation models for total project effort, page authoring effort, media authoring effort, and scripting effort.

12.12 CHECK YOUR PROGRESS

1. Process and project metrics can provide _____ for the generation of quantitative estimate.
2. The "size" of software to be built can be estimated using a direct measure, _____ or an indirect measure, _____.
3. The purpose of planning a project is to identify the sequence of activities as per their complexities and dependencies. (True / False).
4. Cost in a project includes software, hardware, and human resources. (True / False).
5. When there are disagreements between the project lead and the overall project manager, the same can be resolved through ___.
6. COCOMO stands for ___.

Answers to check your progress:

1. Historical perspective and powerful input
2. LOC, FP
3. True
4. True
5. Change control board
6. Constructive Cost Model

12.13 SUMMARY

The software project planner must estimate three things before a project begins: how long it will take, how much effort will be required, and how many people will be involved. In addition, the planner must predict the resources (hardware and software) that will be required and the risk involved. The statement of scope helps the planner to develop estimates using one or more techniques that fall into two broad categories: decomposition and empirical modeling. Decomposition techniques require a delineation of major software functions, followed by estimates of either (1) the number of LOC, (2) selected values within the information domain, (3) the number of person-months required to implement each function, or (4) the number of person-months required for each software engineering activity. Empirical techniques use empirically derived expressions for effort and time to predict these project quantities. Automated tools can be used to implement a specific empirical model. Accurate project estimates generally use at least two of the three techniques just noted. By comparing and reconciling estimates derived using different techniques, the planner is more likely to derive an accurate estimate. Software project estimation can never be an exact science, but a combination of good historical data and systematic techniques can improve estimation accuracy.

12.14 KEYWORDS

- **Software project management**-is an umbrella activity within software engineering. It begins Project complexity, project size, and the degree of structural uncertainty all affect the reliability of estimates.
- **Estimation-** is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

- **Decomposition technique-** is a technique that take a "divide and conquer" approach to software project estimation.
- **Empirical estimation-** is an estimation technique that can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right.

12.15 QUESTIONS FOR SELF STUDY

1. Define software project management. Explain the role of project managers in software project management.
2. What is project estimation? Mention the four steps of software project estimation.
3. List out the factors that affect the project estimation.
4. Briefly explain the project planning process.
5. Discuss the scope of project estimation.
6. Discuss the characteristics and categories of estimation of resources.
7. Write short notes on: (a) LOC (b) FP
8. Give an account on COCOMO Model.
9. Discuss estimation for object oriented projects.

12.16 REFERENCES

1. Software Engineering, A Practitioner's Approach – 7th Edition, Roger S.Pressman
2. Software Project Management in Practice – Pankaj Jalote
3. Software Engineering : Pearson New International Edition – Ian Sommerville, 2013

Karnataka State Open University

Mukthagangothri, Mysore – 570 006.

Dept. of Studies and Research in Management

MBA IT Specialization

IV Semester

MBSC-4.1G Software Project Management

Block 4

PREFACE

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products, . . . the list is almost endless. Software is virtually inescapable in a modern world. And as we move into the twenty-first century, it will become the driver for new advances in everything from elementary education to genetic engineering.

When a computer software succeeds—when it meets the needs of the people who use it, when it performs flawlessly over a long period of time, when it is easy to modify and even easier to use—it can and does change things for the better. But when software fails—when its users are dissatisfied, when it is error prone, when it is difficult to change and even harder to use—bad things can and do happen. We all want to build software that makes things better, avoiding the bad things that lurk in the shadow of failed efforts. To succeed, we need discipline when software is designed and built. We need an engineering approach.

The whole material is organized into four modules each with four units. Each unit lists the objectives of the study along with the relevant questions, illustrations and suggested reading to better understand the concepts.

Wish you happy reading!!!

KARNATAKA STATE OPEN UNIVERSITY

MUKTHAGANGOTRI, MYSURU-06

Dept. of Studies and Research in Management

MBA IT

IV SEMESTER

MBSC-4.1G Software Project Management

BLOCK 4: SOFTWARE MAINTENANCE – REENGINEERING, SOFTWARE PROCES IMPROVEMENT, EMERGING TRENDS

BLOCK 4 INTRODUCTION

Re-engineering is the reorganizing and modifying existing software systems to make them more maintainable. In this block, we are discussing about software maintenance and reengineering techniques. Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

This block consists of four units and is organized as follows:

Unit 13: Software Maintenance – Software Maintenance, Supportability, Reengineering, BPR, Software Reengineering, Reverse Engineering, Restructuring, Forward Engineering, the Economics of Reengineering

Unit 14: Software Process Improvement: SPI Process, the CMMI, People CMM, Other SPI Frameworks, SPI Return on Investment, SPI Trends

Unit 15: **Emerging Trends in Software Engineering:** Technology Evolution, Identifying Soft Trends, Technology Directions, and Tools Related Trends, Agile Project Management

Unit 16: Practicing with an online Project Management Tool

Any one of the tools listed below:

https://www.monday.com

UNIT-13: SOFTWARE MAINTENANCE

STRUCTURE

13.0 OBJECTIVES

After studying this unit, you will be able to:

- Understand about software maintenance
- System re-engineering process.
- Understand Phases of BPR
- Reverse re-engineering process
- Program Restructuring
- Forward Engineering
- Economics of Reengineering

13.1 INTRODUCTION

In this unit, we are going to discuss about software maintenance. The objective of re-engineering is to improve the system structure to make it easier to understand and maintain.

The re-engineering process involves source code translation, reverse engineering, program structure improvement, program modularization and data re-engineering.

13.2 SOFTWARE MAINTENANCE SUPPORTABILITY

Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Need for Maintenance –

Software Maintenance must be performed in order to:

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software.
- Retire software.

Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

Corrective Maintenance - This includes modifications and updating done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

Adaptive Maintenance - This includes modifications and updating applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

Perfective Maintenance - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

Preventive Maintenance - This includes modifications and updating to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Maintenance Activities

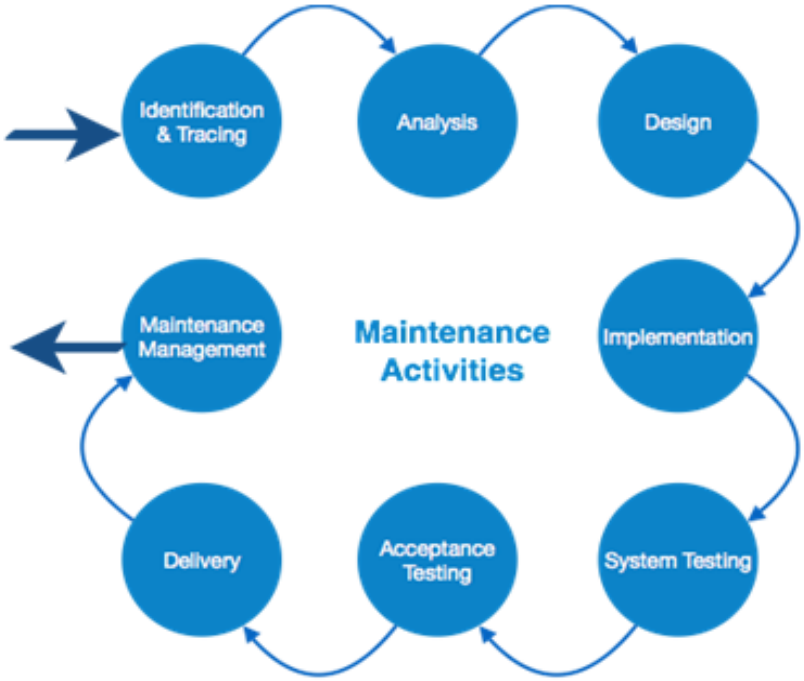


FIGURE 13.1 Maintenance Activities Phase

These activities go hand-in-hand with each of the following phase:

Identification & Tracing - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.

Analysis - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.

Design - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.

I**mplementation** - The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit testing in parallel.

System Testing - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.

Acceptance Testing - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

Delivery - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

Maintenance management - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

13.3 SYSTEM RE-ENGINEERING

Re-structuring or re-writing a part or all of a legacy system without changing its functionality. Applicable where some but not all sub-systems of a larger system require frequent maintenance Re-engineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

When to re-engineer

- When system changes are mostly confined to part of the system then re-engineer that part
- When hardware or software support becomes obsolete
- When new ways of accessing are needed, but its functionality remains
- When tool support is are available

Re-engineering advantages

Reduced risk

There is a high risk in new software development. There may be development problems, staffing problems and specification problems

Reduced cost

The cost of re-engineering is often significantly less than the costs of developing new software.

13.4 BUSINESS PROCESS RE- ENGINEERING (BPR)

Is not just a change, but actually it is a dramatic change and dramatic improvements. This is only achieved through overhaul the organization structures, job descriptions, performance management, training and the most importantly, the use of IT i.e. Information Technology.

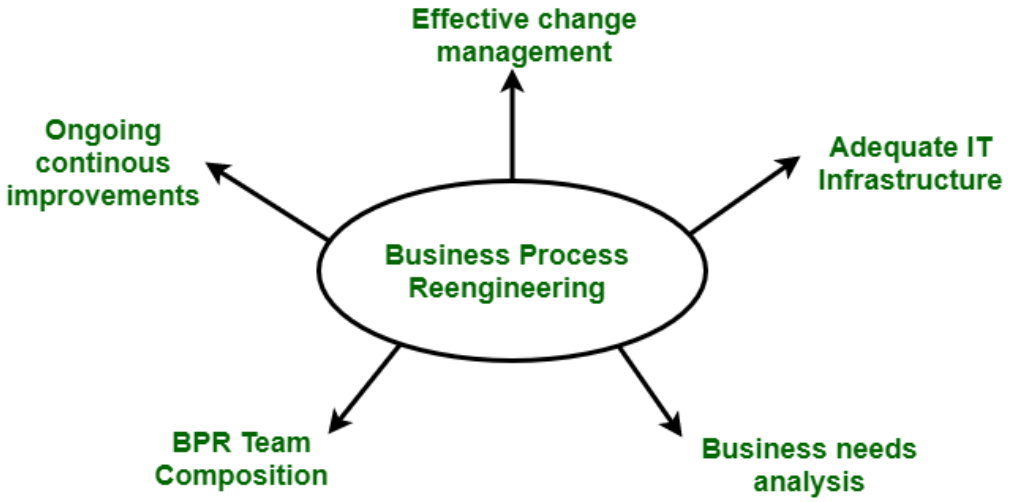


FIGURE 13.2 Business Process Re-engineering

BPR projects have failed sometimes to meet high expectations. Many unsuccessful BPR attempts are due to the confusion surrounding BPR and how it should be performed. It becomes the process of trial and error.

PHASES OF BPR:

According to Peter F. Drucker,**" Re-engineering is new, and it has to be done."** There are 7 different phases for BPR. All the projects for BPR begin with the most critical requirement i.e. communication throughout the organization.

1. Begin organizational change.
2. Build the re-engineering organization.
3. Identify BPR opportunities.
4. Understand the existing process.
5. Reengineer the process

6. Blueprint the new business system.
7. Perform the transformation.

OBJECTIVES OF BPR:

Following are the objectives of the BPR:

- To dramatically reduce cost.
- To reduce time requirements.
- To improve customer services dramatically.
- To reinvent the basic rules of the business e.g. The airline industry.
- Customer satisfaction.
- Organizational learning.

CHALLENGES FACED BY BPR PROCESS:

All the BPR processes are not as successful as described. The companies that have start the use of BPR projects face many of the following challenges:

- Resistance
- Tradition
- Time requirements
- Cost
- Job losses

ADVANTAGES OF BPR:

Following are the advantages of BPR:

- BPR offers tight integration among different modules.
- It offers same views for the business i.e. same database, consistent reporting and analysis.
- It offers process orientation facility i.e. streamline processes.
- It offers rich functionality like templates and reference models.
- It is flexible.
- It is scalable.
- It is expandable.

DISADVANTAGES OF BPR:

Following are the Disadvantages of BPR:

- It depends on various factors like size and availability of resources. So, it will not fit for every business.
- It is not capable of providing an immediate resolution.

13.5 SOFTWARE RE- ENGINEERING

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.

Re-engineering is the reorganizing and modifying existing software systems to make them more maintainable.

Objectives of Re-engineering:

- To describe a cost-effective option for system evolution.
- To describe the activities involved in the software maintenance process.
- To distinguish between software and data re-engineering and to explain the problems of data re-engineering.

Steps involved in Re-engineering:

1. Inventory Analysis
2. Document Reconstruction
3. Reverse Engineering
4. Code Reconstruction
5. Data Reconstruction
6. Forward Engineering

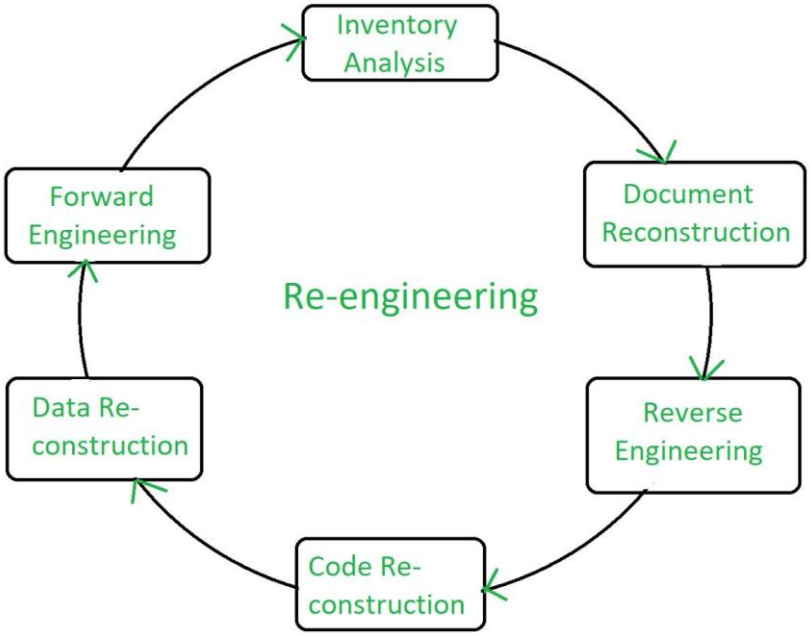


FIGURE 13.3 Steps involved in Re-engineering

Re-engineering Cost Factors:

- The quality of the software to be re-engineered
- The tool support available for re-engineering
- The extent of the required data conversion
- The availability of expert staff for re-engineering

Advantages of Re-engineering:

- **Reduced Risk:** As the software already exists, the risk is less as compared to new software development. Development problems, staffing problems and specification problems are the lots of problems which may arise in new software development.
- **Reduced Cost:** The cost of re-engineering is less than the costs of developing new software.
- **Revelation of Business Rules:** As a system is re-engineered, business rules that are embedded in the system are rediscovered.
- **Better use of Existing Staff:** Existing staff expertise can be maintained and extended accommodate new skills during re-engineering.

Disadvantages of Re-engineering:

- Practical limits to the extent of re-engineering.

- Major architectural changes or radical reorganizing of the systems data management has to be done manually.
- Re-engineered system is not likely to be as maintainable as a new system developed using modern software Re-engineering methods.

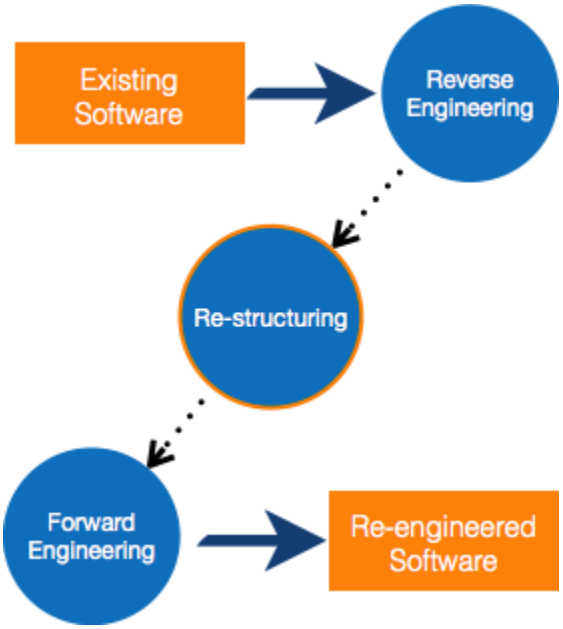


FIGURE 13.4 Steps involved in Re-engineering

RE-ENGINEERING PROCESS

Decide what to re-engineer. Is it whole software or a part of it?

Perform Reverse Engineering, in order to obtain specifications of existing software. Restructure Program if required. Re-structure data as required. Apply Forward engineering concepts in order to get re-engineered software.

13.6 REVERSE ENGINEERING

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, reverse engineering is going in reverse from code to system specification.

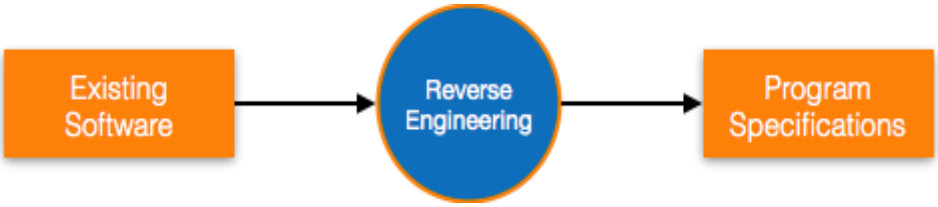


FIGURE 13.5 Steps involved in Reverse engineering

13.7 PROGRAM RESTRUCTURING

It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

13.8 FORWARD ENGINEERING

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.

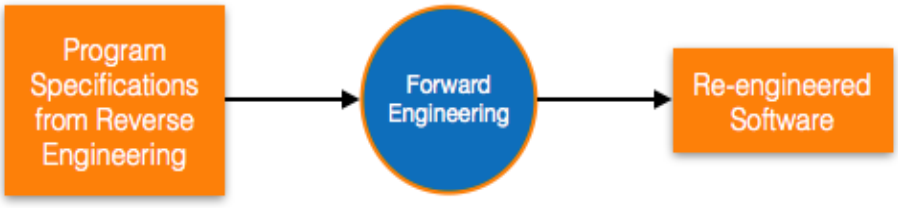


FIGURE 13.6 Steps involved in Forward-engineering

13.9 ECONOMICS OF REENGINEERING:

In a perfect world, every unmaintainable program would be retired immediately; to be replaced by high-quality, reengineered applications developed using modern software engineering practices. But we live in a world of limited resources. Reengineering drains resources that can be used for other business purposes. Therefore, before an organization attempts to reengineer an existing application, it should perform a cost/benefit analysis.

A cost/benefit analysis model for reengineering has been proposed by Sneed. Nine parameters are defined:

P1 = current annual maintenance cost for an application.
P2 = current annual operation cost for an application.
P3 = current annual business value of an application.
P4 = predicted annual maintenance cost after reengineering.
P5 = predicted annual operations cost after reengineering.
P6 = predicted annual business value after reengineering.
P7 = estimated reengineering costs.
P8 = estimated reengineering calendar time.
P9 = reengineering risk factor (P9 = 1.0 is nominal).
L = expected life of the system.

The cost associated with continuing maintenance of a candidate application (i.e., reengineering is not performed) can be defined as

$$Cmaint = [P3 - (P1 + P2)] \times L$$

L The costs associated with reengineering are defined using the following relationship:

$$Creeng = [P6 - (P4 + P5) \times (L - P8) - (P7 \times P9)]$$

Using the costs presented in equations, the overall benefit of reengineering can be computed as

$$\text{cost benefit} = Creeng - Cmaint$$

The cost/benefit analysis presented in the equations can be performed for all high priority applications identified during inventory analysis. Those application that show the highest cost/benefit can be targeted for reengineering, while work on others can be postponed until resources are available.

Cost of maintenance = cost annual of operation and maintenance over application lifetime

Cost of reengineering = predicted return on investment reduced by cost of implementing changes and engineering risk factors

Cost benefit = Cost of reengineering - Cost of maintenance

13.10 CHECK YOUR PROGRESS

1. Maintenance is classified into how many categories?
    a) Two
    b) Four
    c) Five
    d) Three
2. The modification of the software to match changes in the ever changing environment, falls under which category of software maintenance?
    a) Corrective
    b) Adaptive
    c) Perfective
    d) Preventive
3. The process of generating analysis and design documents is known as
    a) Software engineering
    b) Software re-engineering
    c) Reverse engineering
    d) Re-engineering
4. Which of the following is not a business goal of re-engineering?
    a) Cost reduction
    b) Time reduction
    c) Maintainability
    d) None of the mentioned
5. When one does decides to re-engineer a product?
    a) When tools to support restructuring are disabled
    b) When system crashes frequently
    c) When hardware or software support becomes obsolete
    d) Subsystems of a larger system require few maintenance

6. Which of the following is not an example of a business process?
    a) Designing a new product
    b) Hiring an employee
    c) Purchasing services
    d) Testing software
7. Reverse engineering of data focuses on
    a) Internal data structures
    b) Database structures
    c) All of the mentioned
    d) None of the mentioned
8. Reverse engineering techniques for internal program data focus on the definition of classes of object
    a) True b) False

Answers to check your progress:

1. Four
2. Adaptive
3. Reverse engineering
4. None of the mentioned
5. When hardware or software support becomes obsolete
6. Testing software
7. All of the mentioned
8. True

13.11 SUMMARY

The objective of re-engineering is to improve the system structure to make it easier to understand and maintain. The re-engineering process involves source code translation, reverse engineering, program structure improvement, program modularization and data re-engineering. Source code translation is the automatic conversion of of program in one language to another. Reverse engineering is the process of deriving the system design and specification from its source code. Program structure improvement replaces unstructured control constructs with while loops and simple conditionals. Program modularization involves reorganization to group related items.

13.12 KEYWORDS

- **Software maintenance:** It is a process of modifying a software product after its delivery.
- **BPR:** Business Process Re-engineering
- **Software re-engineering:** It is a process where the design of software is changed and re written.
- **Reverse engineering:** It is a process of achieve system specifications by thoroughly analyzing and understanding the existing system.
- Program
- **Program re-structuring:** It is a process to re structure and re constructing the system software.

13.13 QUESTIONS FOR SELF-STUDY

1. What is need maintenance and reengineering of software?
2. What are the 4 types of software maintenance?
3. Write a note on System re-engineering.
4. What is BPR software re-engineering?
5. What are the advantages and disadvantages BPR?
6. Write a note on Software Re-engineering.
7. What are the advantages and disadvantages Software Re-engineering?
8. Explain the following a) Reverse Engineering b)Forward Engineering C) program Restricting
9. Write a note on Economics of Reengineering:

13.14 REFERENCES

1. Software Engineering, A Practitioner's Approach – 7th Edition, Roger S.Pressman
2. Software Project Management in Practice – Pankaj Jalote
3. Software Engineering : Pearson New International Edition – Ian Sommerville, 2013

UNIT-14: SOFTWARE PROCESS IMPROVEMENTS

STRUCTURE

14.0 OBJECTIVES

After studying this unit, you will be able to:

- Explain the SPI process
- Understand the importance of software process improvements
- Study the relation of metrics and SPI
- Understand CMM and CMMI
- Differentiate between CMM and CMMI

14.1 INTRODUCTION

In this unit, we are going to discuss about software process improvements. The term software process improvement (SPI) indicates many things. First, it implies that elements of an effective software process can be defined in an effective manner; second, that an existing organizational approach to software development can be assessed against those elements; and third, that a meaningful strategy for improvement can be defined. The SPI strategy transforms the existing approach to software development into something that is more focused, more repeatable, and more reliable (in terms of the quality of the product produced and the timeliness of delivery). As SPI is not free, it must deliver a return on investment. The effort and time that is required to implement an SPI strategy must pay for itself in some measurable way. To do this, the results of improved process and practice must lead to a reduction in software problems that cost time and money. It must reduce the number of defects that are delivered to end users, reduce the amount of rework due to quality problems, reduce the costs associated with software maintenance and support, and reduce the indirect costs that occur when software is delivered late

Software Process Improvement (SPI) methodology is defined as a series of tasks, tools, and techniques to plan and implement improvement activities to achieve specific goals such as increasing development speed, achieving higher product quality or reducing costs.

Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes. Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

14.2 SPI PROCESS

The difficult part of SPI is not defining the characteristics that define a high-quality software process or the creation of a process maturity model. Those things are comparatively easy than, establishing a consensus for initiating SPI and defining an ongoing strategy for implementing it across a software organization.

The Software Engineering Institute has developed IDEAL— an organizational improvement model that serves as a roadmap for initiating, planning, and implementing improvement actions.

IDEAL is representative of many process models for SPI, defining five distinct activities—

1. initiating
2. Diagnosing
3. Establishing
4. acting, and
5. learning—that guide an organization through SPI activities.

Pressman represents the above activities which applies a commonsense philosophy that requires an organization to (1) look in the mirror, (2) then get smarter so it can make intelligent choices, (3) select the process model (and related technology elements) that best meets its needs, (4) instantiate the model into its operating environment and its culture, and (5) evaluate what has been done. These five activities are applied in an iterative manner in an effort to foster continuous process improvement.

14.2.1 ASSESSMENT AND GAP ANALYSIS

Stated simply, before you begin any journey, it's a good idea to know precisely where you are. The first road-map activity, called assessment, allows you to get your bearings. The intent of assessment is to uncover both strengths and weaknesses in the way the organization applies the existing software process and the software engineering practices that populate the process.

Assessment examines a wide range of actions and tasks that will lead to a high quality process. For instance, regardless of the process model that is chosen, the software organization must establish generic mechanisms such as: defined approaches for customer communication; established methods for representing user requirements; defining a project management framework that includes scoping, estimation, scheduling, and project tracking; risk analysis methods; change management procedures; quality assurance and control activities including reviews; and many others.

Each is considered within the context of the framework that have been established and is assessed to determine whether each of the following questions has been addressed:

- Is the objective of the action clearly defined?
- Are work products required as input and produced as output identified and described?
- Are the work tasks to be performed clearly described?
- Are the people who must perform the action identified by role?
- Have entry and exit criteria been established?
- Have metrics for the action been established?

- Are tools available to support the action?
- Is there an explicit training program that addresses the action?
- Is the action performed uniformly for all projects?

Although the questions noted imply a yes or no answer, the role of assessment is to look behind the answer to determine whether the action in question is being performed in a manner that would conform to best practice.

As the process assessment is conducted, we should also focus on the following attributes:

Consistency: Are important activities, actions, and tasks applied consistently across all software projects and by all software teams?

Sophistication: Are management and technical actions performed with a level of sophistication that implies a thorough understanding of best practice?

Acceptance: Is the software process and software engineering practice widely accepted by management and technical staff?

Commitment: Has management committed the resources required to achieve consistency, sophistication, and acceptance?

The difference between local application and best practice represents a GAP that offers opportunities for improvement. The degree to which consistency, sophistication, acceptance, and commitment are achieved indicates the amount of cultural change that will be required to achieve meaningful improvement.

14.2.2 EDUCATION AND TRAINING

Many practitioners and managers do not know much about either subject. As a consequence, inaccurate perceptions of process and practice lead to inappropriate decisions when an SPI framework is introduced. It follows that a key element of any SPI strategy is education and training for practitioners, technical managers and more senior managers who have direct contact with the software organization. Three types of education and training should be conducted:

1. **Generic concepts and methods** Directed toward both managers and practitioners, this category stresses both process and practice. The intent is to provide professionals with the intellectual tools they need to apply the software process effectively and to make rational decisions about improvements to the process.

2. **Specific technology and tools:** Directed primarily toward practitioners, this category stresses technologies and tools that have been adopted for local use. For example, if UML has been chosen for analysis and design modeling, a training curriculum for software engineering using UML would be established.
3. **Business communication and quality-related topics:** Directed toward all stakeholders, this category focuses on soft topics that help enable better communication among stakeholders and foster a greater quality focus.

In a modern context, education and training can be delivered in a variety of different ways. Everything from podcasts, to Internet-based training, to DVDs, to classroom courses can be offered as part of an SPI strategy.

14.2.3 SELECTION AND JUSTIFICATION

Once the initial assessment activity has been completed and education has begun, a software organization should begin to make choices. These choices occur during a selection and justification activity in which process characteristics and specific software engineering methods and tools are chosen to populate the software process.

First, you should choose the process model that best fits your organization, its stakeholders, and the software that you build. You should decide which of the set of framework activities will be applied, the major work products that will be produced, and the quality assurance checkpoints that will enable your team to assess progress. If the SPI assessment activity indicates specific weaknesses (e.g., no formal SQA functions), you should focus attention on process characteristics that will address these weaknesses directly.

Next, develop a work breakdown for each framework activity (e.g., modeling), defining the task set that would be applied for a typical project. You should also consider the software engineering methods that can be applied to achieve these tasks. As choices are made, education and training should be coordinated to ensure that understanding is reinforced.

Ideally, everyone works together to select various process and technology elements and moves smoothly toward the installation or migration activity. If the criteria for selection are established by committee, people may argue endlessly about whether the criteria are appropriate and whether a choice truly meets the criteria that have been established.

Once a choice is made, time and money must be expended to instantiate it within an

organization, and these resource expenditures should be justified.

14.2.4 INSTALLATION/MIGRATION

Installation is the first point at which a software organization feels the effects of changes implemented as a consequence of the SPI road map. In some cases, an entirely new process is recommended for an organization. Framework activities, software engineering actions, and individual work tasks must be defined and installed as part of a new software engineering culture. Such changes represent a substantial organizational and technological transition and must be managed very carefully.

In other cases, changes associated with SPI are relatively minor, representing small, but meaningful modifications to an existing process model. Such changes are often referred to as process migration.

Installation and migration are actually software process redesign (SPR) activities. When a formal approach to SPR is initiated, three different process models are considered: (1) the existing process, (2) a transitional process, and (3) the target process. If the target process is significantly different from the existing process, the only rational approach to installation is an incremental strategy in which the transitional process is implemented in steps. The transitional process provides a series of way-points that enable the software organization's culture to adapt to small changes over a period of time.

14.2.5 EVALUATION

Although it is listed as the last activity in the SPI road map, evaluation occurs throughout SPI. The evaluation activity assesses the degree to which changes have been instantiated and adopted, the degree to which such changes result in better software quality or other tangible process benefits, and the overall status of the process and the organizational culture as SPI activities proceed.

Both qualitative factors and quantitative metrics are considered during the evaluation activity. From a qualitative point of view, past management and practitioner attitudes about the software process can be compared to attitudes polled after installation of process changes. Quantitative metrics are collected from projects that have used the transitional or "to be" process and compared with similar metrics that were collected for projects that were conducted under the "as is" process.

14.2.6 RISK MANAGEMENT FOR SPI

SPI is a risky undertaking. In fact, more than half of all SPI efforts end in failure. The reasons for failure vary greatly and are organizationally specific. Among the most common risks are: a lack of management support, cultural resistance by technical staff, a poorly planned SPI strategy, an overly formal approach to SPI, selection of an inappropriate process, a lack of buy-in by key stakeholders, an inadequate budget, a lack of staff training, organizational instability, and a myriad of other factors. The role of those chartered with the responsibility for SPI is to analyze likely risks and develop an internal strategy for mitigating them.

A software organization should manage risk at three key points in the SPI process: prior to the initiation of the SPI road map, during the execution of SPI activities (assessment, education, selection, installation), and during the evaluation activity that follows the instantiation of some process characteristic. In general, the following categories can be identified for SPI risk factors: budget and cost, content and deliverables, culture, maintenance of SPI deliverables, mission and goals, organizational management, organizational stability, process stakeholders, schedule for SPI development, SPI development environment, SPI development process, SPI project management, and SPI staff.

Within each category, a number of generic risk factors can be identified. For example, the organizational culture has a strong bearing on risk. The following generic risk factors can be defined for the culture category:

- Attitude toward change, based on prior efforts to change
- Experience with quality programs, level of success
- Action orientation for solving problems versus political struggles
- Use of facts to manage the organization and business
- Patience with change; ability to spend time socializing
- Tools orientation—expectation that tools can solve the problems
- Level of planfulness—ability of organization to plan
- Ability of organization members to participate with various levels of organization openly at meetings
- Ability of organization members to manage meetings effectively
- Level of experience in organization with defined processes

Using the risk factors and generic attributes as a guide, risk exposure is computed in the

following manner:

Exposure = (risk probability) x (estimated loss)

14.2.7 CRITICAL SUCCESS FACTORS

The top five CSFs are presented in this section.

Management commitment and support: Like most activities that precipitate organizational and cultural change, SPI will succeed only if management is actively involved. Senior business managers should recognize the importance of software to their company and be active sponsors of the SPI effort. Technical managers should be heavily involved in the development of the local SPI strategy. Software process improvement is not feasible without investing time, money, and effort. Management commitment and support are essential to sustain that investment.

Staff involvement: SPI cannot be imposed top down, nor can it be imposed from the outside. If SPI efforts are to succeed, improvement must be organic—sponsored by technical managers and senior technologists, and adopted by local practitioners.

Process integration and understanding: The software process does not exist in an organizational vacuum. It must be characterized in a manner that is integrated with other business processes and requirements. To accomplish this, those responsible for the SPI effort must have an intimate knowledge and understanding of other business processes. In addition, they must understand the as is software process and appreciate how much transitional change is tolerable within the local culture.

A customized SPI strategy: There is no cookie-cutter SPI strategy. As I noted earlier in this chapter, the SPI road map must be adapted to the local environment—team culture, product mix, and local strengths and weak nesses must all be considered.

Solid management of the SPI project: It involves coordination, scheduling, parallel tasks, deliverables, adaptation (when risks become realities), politics, budget control, and much more. Without active and effective management, an SPI project is doomed to failure.

SPI mainly consists of 4 cyclic steps as shown in the figure below, while these steps can be broken down into more steps according to the method and techniques used. While in most cases the process will contain these steps.

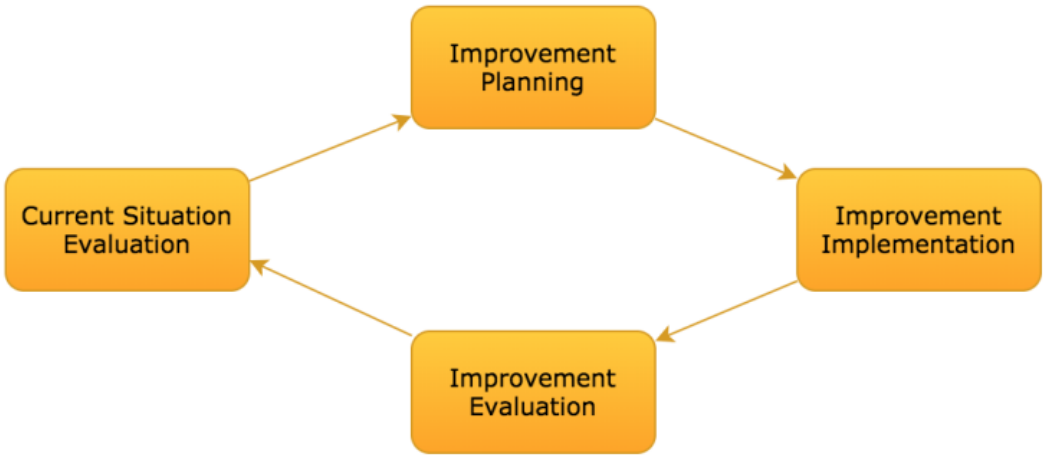


Figure 14.1 Four cycle seps of SPI Process

CURRENT SITUATION EVALUATION

This step is the first step of the process and it is mainly to assess the current situation of the software process by eliciting the requirements from the stakeholders, analyzing the current artifacts and deliverables, and identifying the inefficiencies from the software process. The elicitation can be conducted through different techniques, like, individual interviews, group interview, use-case scenarios, and observations.

The key considerations in this phase, is to identify organization goals and ask the solution-oriented questions. Identifying the proper measurement using, the GQM (Goal – Question – Metric) techniques that will help in measuring the current status and measuring the effectiveness of the improvement process.

IMPROVEMENT PLANNING

After analyzing the current situation and the improvement goals, the findings should be categorized and prioritized according to which one is the most important or have the most severity. We should observe what is the new target level of improvements should look like.

In this phase, the gap between the current level and the target level should be planned in terms of a set of activities to achieve the target. These activities should be prioritized with the alignment of the involved stakeholders and the organization goals, for example, if the project is using the CMMI model, the target could be reaching maturity level 4 and the company at level 3, in that case, the plan should be focused on the process areas and their activities which is related to that level of improvement with the alignment of the organization goal.

IMPROVEMENT IMPLEMENTATION

In the third phase, the planned activities are executed and it puts the improvements into practice and spreads it across the organization, what can be effective at the 2nd, 3rd, and 4th step that planning and implementation could be an iterative way, for example, implementing improvement for improving requirements first, then implementing the reduction for testing process time, and so forth. This iterative way of implementation will help the organization to realize the early benefits from the SPI program early or even adopt the plan if there is no real impact measured from the improvement.

IMPROVEMENT EVALUATION

What is cannot be measured cannot be improved, so in this step, the impact measurement is applied compared with the GQM. Measurement, in general, permits an organization to compare the rate of actual change against its planned change and allocate resources based on the gaps between actual and expected progress.

14.3 PROCESS METRICS AND SOFTWARE PROCESS IMPROVEMENT

One way to improve any process is to measure specific attributes of the process, develop a set of meaningful metrics based on these attributes, and then use the metrics to provide indicators that will lead to a strategy for improvement. Software metrics plays a vital role on software process improvements. So process is one of the controllable factors in improving software quality and organizational performance.

The personal software process (PSP) is a structured set of process descriptions, measurements, and methods that can help engineers to improve their personal performance. It provides the forms, scripts, and standards that help them estimate and plan their work. It shows them how to define processes and how to measure their quality and productivity. A fundamental PSP principle is that everyone is different and that a method that is effective for one engineer may not be suitable for another. The PSP thus helps engineers to measure and track their own work so they can find the methods that are best for them.

As an organization becomes more comfortable with the collection and use of process metrics, the derivation of simple indicators gives way to a more rigorous approach called statistical software process improvement (SSPI). In essence, SSPI uses software failure analysis to collect information about all errors and defects encountered as an application, system, or

product is developed and used. Failure analysis works in the following manner:

1. All errors and defects are categorized by origin (e.g., flaw in specification, flaw in logic, nonconformance to standards).
2. The cost to correct each error and defect is recorded
3. The number of errors and defects in each category is counted and ranked in descending order.
4. The overall cost of errors and defects in each category is computed.
5. Resultant data are analyzed to uncover the categories that result in highest cost to the organization.
6. Plans are developed to modify the process with the intent of eliminating (or reducing the frequency of) the class of errors and defects that is most costly.

14.4 THE CMM (Capability Maturity Model)

Capability Maturity Model was developed by the Software Engineering Institute (SEI), a research and development center sponsored by the U.S. Department of Defense (DOD) and now part of Carnegie Mellon University. SEI was founded in 1984 to address software engineering issues and, in a broad sense, to advance software engineering methodologies. More specifically, SEI was established to optimize the process of developing, acquiring and maintaining heavily software-reliant systems for the DOD. SEI advocates industry-wide adoption of the CMM Integration (CMMI), which is an evolution of CMM. The CMM model is still widely used as well.

CMM is similar to ISO 9001, one of the ISO 9000 series of standards specified by the International Organization for Standardization. The ISO 9000 standards specify an effective quality system for manufacturing and service industries; ISO 9001 deals specifically with software development and maintenance.

The Capability Maturity Model is a method used to develop and process an organization's software development process. The CMM model has five-level evolutionary path of increasingly organized and systematically more mature processes.

- CMM is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.
- It is based on profound feedback and development practices adopted by the most

successful organizations worldwide.

- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

Shortcomings of SEI/CMM:

It encourages the achievement of a higher maturity level in some cases by displacing the true mission, which is improving the process and overall software quality. It only helps if it is put into place early in the software development process. CMM has no formal theoretical basis and in fact is based on the experience of very knowledgeable people. It does not have good empirical support and this same empirical support could also be constructed to support other models.

Key Process Areas (KPA's):

Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity. Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.

The five levels of CMM are as follows:

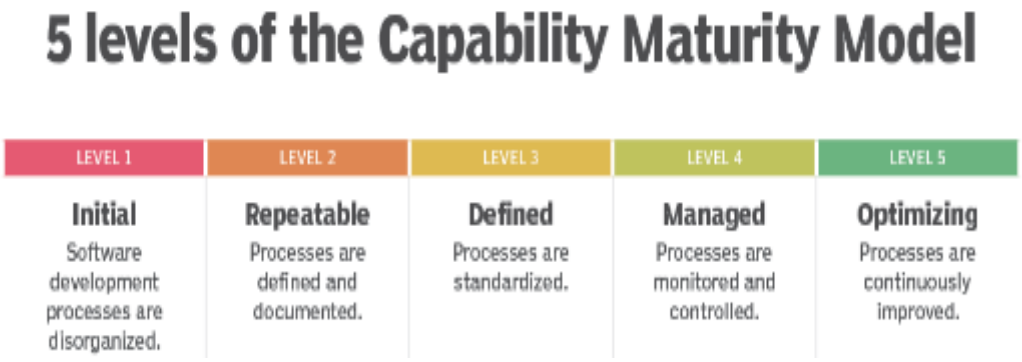


Figure 14.2 Five levels of CMM

Level-1: Initial –

- No KPA's defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.

- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- **Project Planning-** It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management-** It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
- **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.

Level-3: Defined –

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- **Peer Reviews-** In this method, defects is removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.
- **Organization Process Definition-** Its key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.

- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- **Software Quality Management-** It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.
- **Process Change Management-** Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.
- **Technology Change Management-** It consists of the identification and use of new technologies to improve product quality and decrease product development time.
- **Defect Prevention-** It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

14.5 THE CMMI

Capability Maturity Model Integration (CMMI) is a successor of CMM and is a more evolved model that incorporates best components of individual disciplines of CMM like Software CMM, Systems Engineering CMM, People CMM, etc. Since CMM is a reference model of matured practices in a specific discipline, so it becomes difficult to integrate these disciplines as per the requirements. This is why CMMI is used as it allows the integration of multiple disciplines as and when needed.

Objectives of CMMI:

- Fulfilling customer needs and expectations.
- Value creation for investors/stockholders.
- Market growth is increased.
- Improved quality of products and services.
- Enhanced reputation in Industry.

CMMI Representation

CMMI has two types of representation, staged representation and continuous representation.

1. **Staged Representation:**
   - Staged representation uses a pre-defined set of process areas to define improvement path.
   - It provides a sequence of improvements, where each part in the sequence serves as a foundation for the next.
   - It is an improved path is defined by maturity level.
   - The maturity level describes the maturity of processes in organization.
   - Staged CMMI representation allows comparison between different organizations for multiple maturity levels.

2. **Continuous Representation**
   - Allows selection of specific process areas.
   - Continuous representation uses capability levels that measures improvement of an individual process area.
   - Continuous CMMI representation allows comparison between different organizations on a process-area-by-process-area basis.
   - It allows organizations to select processes which require more improvement.
   - In this representation, order of improvement of various processes can be selected which allows the organizations to meet their objectives and eliminate risks.

CMMI MODEL – MATURITY LEVELS:

In CMMI with staged representation, there are five maturity levels described as follows:

Maturity level 1: Initial

- Processes are poorly managed or controlled.
- Unpredictable outcomes of processes involved.
- Ad hoc and chaotic approach used.
- No KPAs (Key Process Areas) defined.
- Lowest quality and highest risk.

Maturity level 2: Managed

- At this level requirements are managed.
- Processes are planned and controlled.

- Projects are managed and implemented according to their documented plans.
- This risk involved is lower than Initial level, but still exists.
- Quality is better than Initial level.

Maturity level 3: Defined

- In the third level, processes are well characterized and described using standards, proper procedures, and methods, tools, etc.
- Medium quality and medium risk involved.
- Focus is process standardization.

Maturity level 4: Quantitatively managed

- At this level quantitative objectives for process performance and quality are set.
- Quantitative objectives are based on customer requirements, organization needs, etc.
- Process performance measures are analyzed quantitatively.
- Higher quality of processes is achieved.
- lower risk

Maturity level 5: Optimizing

- Continuous improvement in processes and their performance.
- Improvement has to be both incremental and innovative.
- Highest quality of processes.
- Lowest risk in processes and their performance.

CMMI MODEL – CAPABILITY LEVELS

A capability level includes relevant specific and generic practices for a specific process area that can improve the organization's processes associated with that process area. For CMMI models with continuous representation, there are six capability levels as explained below:

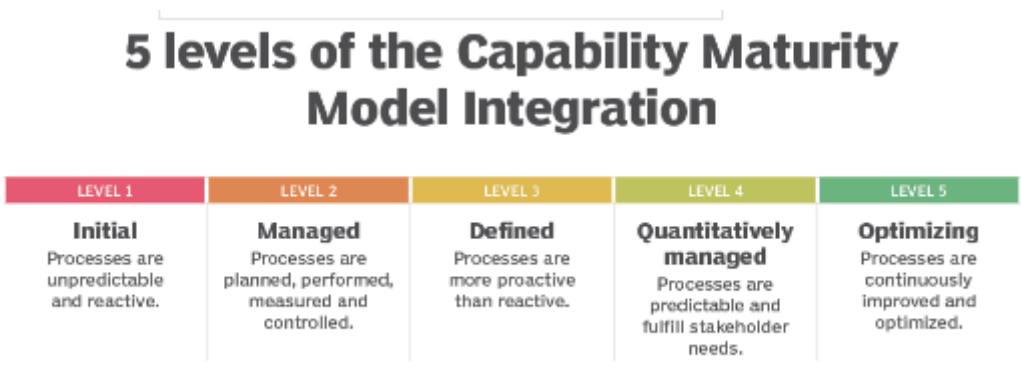


Figure 14.3 Five levels of CMMI

Capability level 0: Incomplete

- Incomplete process – partially or not performed.
- One or more specific goals of process area are not met.
- No generic goals are specified for this level.
- This capability level is same as maturity level 1.

Capability level 1: Performed

- Process performance may not be stable.
- Objectives of quality cost and schedule may not be met.
- A capability level 1 process is expected to perform all specific and generic practices for this level.
- Only a start-step for process improvement.

Capability level 2: Managed

- Process is planned, monitored and controlled.
- Managing the process by ensuring that objectives are achieved.
- Objectives are model and other including cost, quality, schedule.
- Actively managing processing with the help of metrics.

Capability level 3: Defined

- A defined process is managed and meets the organization's set of guidelines and standards.
- Focus is process standardization.

Capability level 4: Quantitatively Managed

- Process is controlled using statistical and quantitative techniques.
- Process performance and quality is understood in statistical terms and metrics.
- Quantitative objectives for process quality and performance are established.

Capability level 5: Optimizing

- Focuses on continually improving process performance.
- Performance is improved in both ways – incremental and innovation.
- Emphasizes on studying the performance results across the organization to ensure that common causes or issues are identified and fixed.

14.6 SPI RETURN ON INVESTMENT

Many organizations and firms invest in software process improvement (SPI). This is to be done in order to satisfy business goals for customer satisfaction, time-to market, cost, quality, and reliability. Return on investment (ROI) is a traditional approach for measuring the business or monetary value of an investment.

It is important to note that ROI is a metric that can be used before and after an investment in SPI. ROI can be used to evaluate (a priori) investment opportunities and make a proper selection and ROI can be used to evaluate (a posteriori) the extent to which an investment was legitimate.

ROI numbers are however useful as they can be used to:

• Convince strategic stakeholders to invest money and effort into SPI, and to convince them that organizational performance issues can be solved through SPI.

• Estimate the amount of effort necessary to solve a certain problem, or estimating whether a certain intended benefit is worth its cost.

• Decide how to prioritize software process improvements and which software processes to improve first, as many organizations have severe timing and resource constraints.

• Decide whether to continue SPI initiatives and programs. SPI budgets are assigned and discussed yearly, so if benefits are not made explicit and a sufficient ROI is not shown, continuation is at risk.

• Simply survive, as any investment in an organization should be valued against its return. Or else, it is very likely that money will be wasted and that there is risk of bankruptcy in the long run.

ROI METRICS

ROI metrics are designed to measure the economic value of a new and improved software process. Each ROI metric is a relevant indicator of how much a new and improved software process is worth. We recommend only six basic metrics related to ROI, as shown in Figure 11.3. They are costs, benefits, benefit/cost ratio or B/CR, return on investment or ROI, net present value or NPV, and break-even point or BEP.

ROI METRICS SHOWING SIMPLICITY OF RETURN ON INVESTMENT FORMULAS AND THEIR ORDER OF APPLICATION

Metric	Definition	Formula
Costs	Total amount of money spent on a new and improved software process	$\sum_{i=1}^{n} Cost_i$
Benefits	Total amount of money gained from a new and improved software process	$\sum_{i=1}^{n} Benefit_i$
B/CR	Ratio of benefits to costs	$\frac{Benefits}{Costs}$
ROI	Ratio of adjusted benefits to costs	$\frac{Benefits - Costs}{Costs} \times 100\%$
NPV	Discounted cash flows	$\sum_{i=1}^{Years} \frac{Benefits_i}{(1 + Discount\ Rate)^{Years}} - Costs_0$
BEP	Point when benefits meet or exceed cost	$\frac{Costs}{Old\ Costs / New\ Costs - 1}$

Figure 14.4 ROI metrics

Each ROI metric builds upon its predecessor and refines the accuracy of the economic value of a new software process. ROI metrics are not necessarily independent or mutually exclusive. Each ROI metric must be considered individually. For example, costs may be astronomical or benefits may be negligible, marginalizing the relevance of the other metrics. Costs consist of the amount of money an organization has to pay in order to implement a SPI method. Benefits generally consist of the amount of money saved by implementing a SPI method. B/CR is a simple ratio of the amount of money saved implementing a new SPI method to the amount of money consumed. ROI is also a ratio of money saved to money consumed by a new SPI method expressed as a percentage. However, the ROI metric demands that the costs of implementing the SPI method must first be subtracted from the benefits.

14.7 PEOPLE CMM

A software process, no matter how well conceived, will not succeed without talented, motivated software people. The People Capability Maturity Model is a roadmap for implementing workforce practices that continuously improve the capability of an organization's workforce. Developed in the mid-1990s and refined over the intervening years, the goal of the People CMM is to encourage continuous improvement of generic workforce knowledge, specific software engineering and project management skills, and process-related abilities.

Like the CMM, CMMI, and related SPI frameworks, the People CMM defines a set of five organizational maturity levels that provide an indication of the relative sophistication of workforce practices and processes. These maturity levels are tied to the existence within an

organization of a set of key process areas (KPAs). An overview of organizational levels and related KPAs is shown in

Figure 14.1. People CMM complements any SPI framework by encouraging an organization to nurture and improve its most important asset—its people. As important, it establishes a workforce atmosphere that enables a software organization to attract, develop, and retain outstanding talent.

Level	Focus	Process Areas
Optimized	*Continuous improvement*	Continuous workforce innovation Organizational performance alignment Continuous capability improvement
Predictable	*Quantifies and manages knowledge, skills, and abilities*	Mentoring Organizational capability management Quantitative performance management Competency-based assets Empowered workgroups Competency integration
Defined	*Identifies and develops knowledge, skills, and abilities*	Participatory culture Workgroup development Competency-based practices Career development Competency development Workforce planning Competency analysis
Managed	*Repeatable, basic people management practices*	Compensation Training and development Performance management Work environment Communication and co-ordination Staffing
Initial	*Inconsistent practices*	

FIGURE 14.5 Process areas for the People CMM

14.8 OTHER SPI FRAMEWORKS

SEI's CMM and CMMI are the most widely applied SPI frameworks; there are other alternatives frameworks have been proposed and are in use. Among the most widely used of these alternatives are:

• **SPICE (Software Process Improvement and Capability dEtermination) —**an international initiative to support ISO's process assessment and life cycle process standards. The SPICE model provides an SPI assessment framework that is compliant with ISO 15504:2003 and ISO 12207. The SPICE document suite presents a complete SPI framework including a model for

process management, guidelines for conducting an assessment and rating the process under consideration, construction, selection, and use of assessment instruments and tools, and training for assessors.

• **ISO/IEC 15504** for (Software) Process Assessment.

• **Bootstrap—**an SPI framework for small and medium-sized organizations that conforms to SPICE. The Bootstrap SPI framework has been developed to ensure conformance with the emerging ISO standard for software process assessment and improvement (SPICE) and to align the methodology with ISO 12207. The objective of Bootstrap is to evaluate a software process using a set of software engineering best practices as a basis for assessment. Like the CMMI, Bootstrap provides a process maturity level using the results of questionnaires that gather information about the as is software process and software projects. SPI guidelines are based on maturity level and organizational goals.

• **PSP and TSP—**individual and team-specific SPI frameworks, which focus on process in-the-small, a more rigorous approach to software development coupled with measurement. Both PSP and TSP emphasize the need to continuously collect data about the work that is being performed and to use that data to develop strategies for improvement.

• **TickIT—**an auditing method that assesses an organization's compliance to ISO Standard 9001:2000. The Ticket auditing method ensures compliance with ISO 9001:2000 for Software—a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

14.9 SPI TRENDS

Over the past two decades, many companies have attempted to improve their software engineering practices by applying an SPI framework to effect organizational change and technology transition.

David Rico reports that a typical application of an SPI framework such as the SEI CMM can cost between $25,000 and $70,000 per person and take years to complete! It should come as no surprise that the future of SPI should emphasize a less costly and time-consuming approach. To be effective in the twenty-first century world of software development, future SPI frameworks must become significantly more agile. Rather than an organizational focus, which can take years to complete successfully, contemporary SPI efforts should focus on the project level, working to

improve a team process in weeks, not months or years. To achieve meaningful results (even at the project level) in a short time frame, complex framework models may give way to simpler models. Any attempt at SPI demands a knowledgeable workforce, but education and training expenses can be expensive and should be minimized and streamlined. Future SPI efforts should rely on Web-based training that is targeted at pivotal practices. The frameworks and models that have been developed represent substantial intellectual assets for the software engineering community. But like all things, these assets guide future attempts at SPI not by becoming a recurring dogma, but by serving as the basis for better, simpler, and more agile SPI models.

14.10 CHECK YOUR PROGRESS

1. What are the goals of SPI?
2. Understanding existing process and changing these process to increase quality and reduce cost and development time is called_____.
3. _____ helps the engineers to measure and track their own work.
4. The CMM model has ___ level of evolutionary path.
5. ____is a metric that can be used before and after an investment in SPI.

Answers to check your progress

1. i. Increase development speed
   ii. Achieve high quality product
   iii. Reduce cost
2. Process improvement
3. Personal software process (PSP)
4. Five
5. ROI

14.11 SUMMARY

A software process improvement framework defines the characteristics that must be present if an effective software process is to be achieved, an assessment method that helps determine whether those characteristics are present, and a strategy for assisting a software organization in implementing those process characteristics that have been found to be weak or missing. Regardless of the constituency that sponsors SPI, the goal is to improve process quality and, as a consequence, improve software quality and timeliness. A process maturity model

provides an overall indication of the process maturity exhibited by a software organization. It provides a qualitative feel for the relative effectiveness of the software process that is currently being used.

The SPI road map begins with assessment— a series of evaluation activities that uncover both strengths and weaknesses in the way the organization applies the existing software process and the software engineering practices that populate the process. As a consequence of assessment, a software organization can develop an overall SPI plan. One of the key elements of any SPI plan is education and training, an activity that focuses on improving the knowledge level of managers and practitioners. Once staff becomes well versed in current software technologies, selection and justification commence. These tasks lead to choices about the architecture of the software process, the methods that populate it, and the tools that support it. Installation and evaluation are SPI activities that instantiate process changes and assess their efficacy and impact.

To successfully improve its software process, an organization must exhibit the following characteristics: management commitment and support for SPI, staff involvement throughout the SPI process, process integration into the overall organizational culture, an SPI strategy that has been customized for local needs, and solid management of the SPI project. A number of SPI frameworks are in use today. The SEI's CMM and CMMI are widely used. The People CMM has been customized to assess the quality of the organizational culture and the people who populate it. SPICE, Bootstrap, PSP, TSP, and TickIT are additional frameworks that can lead to effective SPI.

SPI is hard work that requires substantial investment of dollars and people. To ensure that a reasonable return on investment is achieved, an organization must measure the costs associated with SPI and the benefits that can be directly attributed to it.

14.12 KEYWORDS

- **SPI:** Is a methodology defined as a series of tasks, tools and techniques to implement improvement activities.
- **SSPI:** Statistical Software Process Improvement.
- **SEI:** Software Engineering Institute is a research and develop center.
- **CMM:** Is a method used to develop and process organization software develops process.
- **CMMI:** Is a successor of CMM.
- **ROI:** Is a traditional approach for measuring the business or monetary of an ROI.

14.13 QUESTIONS FOR SELF STUDY

1. Define SPI in your own words?
2. What is process improvement?
3. Discuss SPI process.
4. Explain assessment and gap analysis.
5. Elaborate the role of education and training in the SPI.
6. Discuss the risk management for SPI.
7. What are the four cycles of SPI process and explain each stage.
8. What is SSPI? How it is used?
9. Discuss CMM model.
10. What is CMMI? What are its types of representation?
11. Discuss ROI metric.
12. Examine other SPI frameworks.
13. Review people CMM.
14. Explain SPI trends.

14.14 REFERENCES

1. Software Engineering, A Practitioner's Approach – 7th Edition, Roger S.Pressman
2. Software Project Management in Practice – Pankaj Jalote
3. Software Engineering : Pearson New International Edition – Ian Sommerville, 2013

UNIT-15: EMERGING TRENDS IN SOFTWARE ENGINEERING

STRUCTURE

15.0 OBJECTIVES

After studying this unit, you will be able to:

- Discuss about Emergent requirements and why do they present a challenge to software engineers
- Explain different stages in a technology's life cycle
- Identify soft and hard trends.
- Describe how to Manage the Complexity
- Discuss about technology evolution and its directions
- Describe model driven and test driven software

15.1 INTRODUCTION

In this unit, we are going to discuss about emerging trends in software engineering. Unlike 'Engineering industries', software industry is about 50 years old, practitioners and researchers have developed an array of process models, technical methods, and automated tools in an effort to foster fundamental change in the way we build computer software. However, past experience indicates that there is a tacit desire to find the "silver bullet" the magic process or transcendent technology that will allow us to build large, complex,

software-based systems easily, without confusion, without mistakes, without delay, without many problems. No one can predict the future with absolute certainty, but it is possible to assess trends in the software engineering area and from those trends to suggest possible directions for the technology.

15.2 TECHNOLOGY EVOLUTION

Ray Kurzweil argues that technological evolution is similar to biological evolution, but occurs at a rate that is orders of magnitude faster. Evolution (whether biological or technological) occurs as a result of positive feedback—"the more capable methods resulting from one stage of evolutionary progress are used to create the next stage". The big questions for the twenty-first century are: (1) how rapidly does a technology evolve? (2) How significant are the effects of positive feedback. (3) How profound will the resultant changes be?

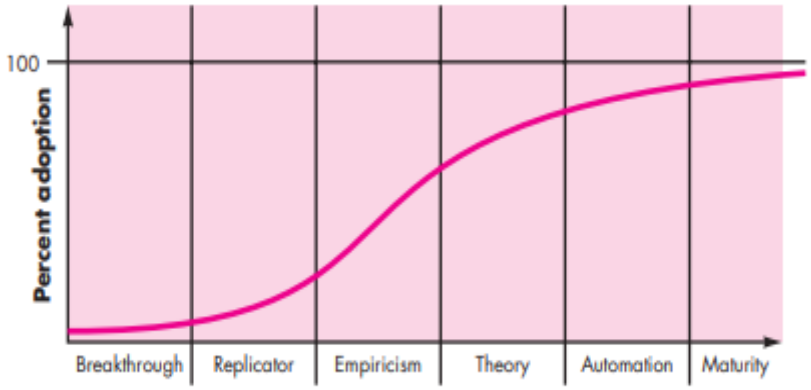


Fig 15.2.1: A technology innovation life cycle

When a successful new technology is introduced, the initial concept moves through a reasonably predictable "innovation life cycle" illustrated in Figure 15.2.1.

- In the breakthrough phase, a problem is recognized and repeated attempts at a viable solution are attempted. At some point, a solution shows promise.
- The initial breakthrough work is reproduced in the replicator phase and gains wider usage.
- Empiricism leads to the creation of empirical rules that govern the use of the technology
- Repeated success leads to a broader theory of usage
- Then creation of automated tools during the automation phase.
- Finally, the technology matures and is used widely.

Many research and technology trends never reach maturity. In fact, the vast majority of "promising" technologies in the software engineering domain receives widespread interest for a few years and then fall into niche usage by a dedicated band of adherents. This is not to say that these technologies lack merit, but rather to emphasize that the journey through the innovation life cycle is long and hard.

Computing technologies evolve through an "S-curve" that exhibits

- relatively slow growth during the technology's formative years
- rapid acceleration during its growth period
- and then a leveling-off period as the technology reaches its limits.

But computing and other related technologies have exhibited explosive (exponential) growth during the central stages shown in Figure 15.2.1 and will continue to do so.

Today, we are at the knee of the S-curve for modern computing technologies—at the transition between early growth and the explosive growth that is to follow. The implication is that over the next 20 to 40 years, we will see dramatic (even mind-boggling) changes in computing capability. The coming decades will result in order-of-magnitude changes in computing speed, size, capacity, and energy consumption (to name only a few characteristics).

Within 20 years, technology evolution will accelerate at an increasingly rapid pace, ultimately leading to an era of non-biological intelligence that will merge with and extend human intelligence in ways that are fascinating to contemplate.

By the year 2040, a combination of extreme computation, nanotechnology, massively high bandwidth ubiquitous networks, and robotics will lead us into a different world.

OBSERVING SOFTWARE ENGINEERING TRENDS

Software engineers face the often formidable challenges of dealing with rapid change, uncertainty and emergence, dependability, diversity, and interdependence, but they also have opportunities to make significant contributions that will make a difference for the better.

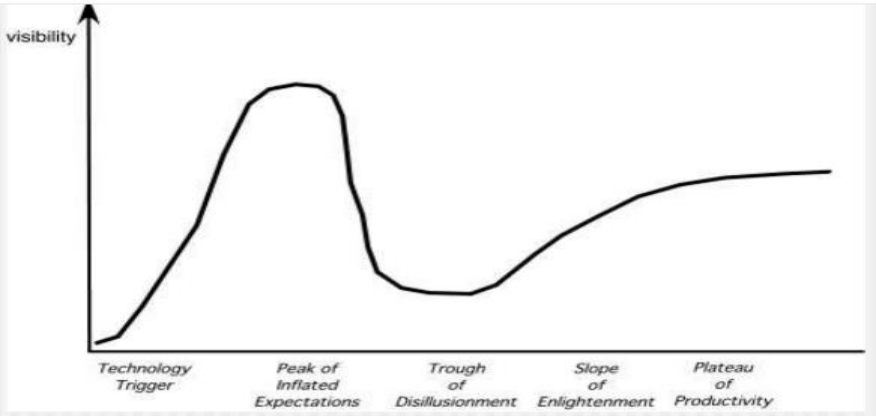
The "soft trends" have a significant impact on the overall direction of software engineering. But other ("harder") research and technology-oriented trends remain important. Technology trends occur when research trends are extrapolated to meet industry needs and are shaped by market-driven demand.

A consultancy that studies technology trends across many industries—has developed a hype cycle for emerging technologies, represented in Fig 15.3.1.

The Hype Cycle: The hype cycle is **a graphical representation of the life cycle stages a technology goes through from conception to maturity and widespread adoption**. The hype cycle is a branded tool created by Gartner, an information technology (IT) research and Consultancy Company.

However, the hype cycle's stages are often used as reference points in marketing and technology reporting. Businesses can use the hype cycle to guide technology decisions in accordance with their level of comfort with risk. Each stage of the cycle is associated with its own risks and opportunities. The hype cycle identifies five overlapping stages in a technology's life cycle as shown in Fig 15.3.1

1. **Technology Trigger**: In this stage, a technology is conceptualized. There may be prototypes but there are often no functional products or market studies. The potential spurs media interest and sometimes proof-of-concept demonstrations are available at this stage.
2. **Peak of Inflated Expectations**: The technology is implemented, especially by early adopters. There is a lot of publicity about both successful and unsuccessful implementations.
3. **Trough of Disillusionment**: Flaws and failures lead to some disappointment in the technology. Some producers are unsuccessful or drop their products. Continued investments in other producers are contingent upon addressing problems successfully.
4. **Slope of Enlightenment**: The technology's potential for further applications becomes more broadly understood and an increasing number of companies implement or test it in their environments. Some producers create further generations of products.
5. **Plateau of Productivity**: The technology becomes widely implemented; its place in the market and its applications are well-understood. Standards arise for evaluating technology providers.



15.3.1: Various stages of Hype cycle

15.3 IDENTIFYING SOFT TRENDS

Each nation with a substantial IT industry has a set of unique characteristics that define the manner in which business is conducted, the organizational dynamics that arise within a company, the distinct marketing issues that apply to local customers, and the overriding culture that dictates all human interaction. However, some trends in each of these areas are universal and have as much to do with sociology, anthropology, and group psychology (often referred to as the "soft sciences") as they do with academic or industrial research. Hard trends are the technical aspects of next generation software intensive systems. It shows the technical directions that software engineering process, methods and tools will take.

a) Connectivity and collaboration (enabled by high bandwidth communication) has already led to a software team that does not occupy the same physical space (telecommuting and part-time employment in a local context). One team collaborates with other teams that are separated by time zones, primary language, and culture. Software engineering must respond with an overarching process model for "distributed teams" that is agile enough to meet the demands of immediacy but disciplined enough to coordinate disparate groups.

b) Globalization leads to a diverse workforce (in terms of language, culture, problem resolution, management philosophy, communication priorities, and person-to-person interaction). Different teams (in different countries) must respond to engineering problems in a way that best accommodates their unique needs, while at the same time fostering a level of uniformity that allows a global project to proceed. This type of organization suggests fewer levels of management and a greater emphasis on team-level decision making. It can lead to greater agility, but only if communication mechanisms have been established so that every team can understand project and technical status (via networked groupware) at any time.

c) An aging population implies that many experienced software engineers and managers will be leaving the field over the coming decade. Viable mechanisms are needed that can capture the knowledge of these aging managers and technologies. In other regions of the world, the number of young people available to the software industry is exploding. This provides an opportunity to mold a software engineering culture without the burden of 50 years of "old-school" prejudices.

d) Consumer spending in emerging economies will double to well over $9 trillion. Some of this spending will be applied to products and services that have a digital component that are software-based or software-driven.

e) People and Teams

i) As systems grow in size, teams grow in number, geographical distribution, and culture.

ii) As systems grow in complexity, team interfaces become pivotal to success.

iii) As systems become pervasive, teams must manage emergent requirements.

iv) As systems become more open, what is a team?

Finally, human culture itself will impact the direction of software engineering. Every generation establishes its own imprint on local culture, and yours will be no different. According to well-known consultant who specializes in cultural trends, characterizes them in the following manner: "Our Trends are not fads. Our Trends endure. Our Trends evolve. They represent underlying forces, first causes, basic human needs, attitudes, aspirations. They help us navigate the world, understand what's happening and why, and prepare for what is yet to come." A detailed discussion of how modern cultural trends will have an impact on software engineering is best left to those who specialize in the "soft sciences."

15.3.1 MANAGING COMPLEXITY

In the relatively near future, systems requiring over 1 billion lines of source code (LOC) will begin to emerge.

a) Consider the interfaces for a billion LOC system to the outside world,to other interoperable systems, to the Internet (or its successor), and to the millions of internal components that must all work together to make this computing monster operate successfully. Is there a reliable way to ensure that all of these connections will allow information to flow properly?

b) Consider the number of people (and their locations) who will be doing the work, the coordination of people and technology, the unrelenting flow of changes, the likelihood of a multiplatform, multi operating system environment. Is there a way to manage and coordinate people who are working on a monster project?

c) Consider the engineering challenge. How can we analyze tens of thousands of requirements, constraints, and restrictions in a way that ensures that inconsistency and ambiguity, omissions, and outright errors are uncovered and corrected? How can we create a design architecture that is robust enough to handle a system of this size? How can software

engineers establish a change management system that will have to handle hundreds of thousands of changes?

d) Consider the challenge of quality assurance. How can we perform verification and validation in a meaningful way? How do you test a 1 billion LOC system?

In the early days, software engineers attempted to manage complexity in what can only be described as an ad hoc fashion. Today, we use process, methods, and tools to keep complexity under control.

15.3.2 OPEN-WORLD SOFTWARE

- Software that is designed to adapt to a continually changing environment 'by self-organizing its structure and self-adapting its behavior.
- Concepts such as ambient intelligence, context-aware applications, and pervasive/ubiquitous computing all focus on integrating software-based systems into an environment far broader than anything to date.
- Today's pervasive computing contains device mobility and ad hoc networking and simple context awareness. Soon smart objects implemented in devices that have the potential to communicate with one another also evolve.
- Over the next decade, mobile user profiles that can be recognized by other objects and smart objects will respond to other objects based on situational characteristic.

It should be obvious that significant privacy and security issues come into play. A "trust management system" will manage privileges that enable communication with networks, health, entertainment, financial, employment, and personal systems.

New capable systems will be added to the network constantly, each providing useful capabilities and demanding access to your P-com. Therefore, the P-com software must be designed so that it can adapt to the requirements that emerge.

15.3.3 EMERGENT REQUIREMENTS

Requirements will emerge as everyone involved in the engineering and construction of a complex system learns more about it, the environment in which it is to reside, and the users who will interact with it.

- First, process models must be designed to embrace change and adopt the basic tenets of the agile philosophy.

- Next, methods that yield engineering models (e.g., requirements and design models) must be used judiciously because those models will change repeatedly as more knowledge about the system is acquired.
- Finally, tools that support both process and methods must make adaptation and change easy.

As the number of changes grows, the likelihood of unintended side effects also grows. This should be a cause for concern as complex systems with emergent requirements become the norm.

15.3.4 THE TALENT MIX

Each software team must bring a variety of creative talent and technical skills to its part of a complex system, and the overall process must allow the output of these islands of talent to merge effectively.

15.3.5 SOFTWARE BUILDING BLOCKS

All of us who have fostered a software engineering philosophy have emphasized the need for reuse—of source code, object-oriented classes, components, patterns, and COTS software. Although the software engineering community has made progress as it attempts to capture past knowledge and reuse proven solutions, a significant percentage of the software that is built today continues to be built "from scratch." Part of the reason for this is a continuing desire (by stakeholders and software engineering practitioners) for "unique solutions."

In the hardware world, original equipment manufacturers (OEMs) of digital devices use application-specific standard products (ASSPs) produced by silicon vendors almost exclusively.

One advantage of the use of software components is that the OEM can leverage the functionality provided by the software without having to develop in-house expertise in the specific functions or invest developer time on the effort to implement and validate the components. Other advantages include the ability to acquire and deploy only the specific set of functionalities that are needed for the system, as well as the ability to integrate these components into an already-existing architecture.

However, the software component approach does have a disadvantage in that there is a given level of effort required to integrate the individual components into the overall product. This integration challenge may be further complicated if the components are sourced from a variety of vendors, each with its own interface methodologies. As additional sources of

components are used, the effort required managing various vendors' increases, and there is a greater risk of encountering problems related to the interaction across components from different sources.

15.3.6 CHANGING PERCEPTIONS OF "VALUE"

When computer software is considered, the modern perception of value is changing from business value (cost and profitability) to customer values that include: speed of delivery, richness of functionality, and overall product quality.

15.3.7 OPEN SOURCE

Open source is a development method for software that harnesses the power of distributed peer review and transparency of process.

The advantages of open source is

- better quality,
- higher reliability,
- more flexibility,
- lower cost,
- An end to predatory vendor lock-in.

The term open source when applied to computer software, implies that software engineering work products (models, source code, test suites) are open to the public and can be reviewed and extended (with controls) by anyone with interest and permission.

An open-source "team" may have a number of full-time "dream team" members, but the number of people working on the software expands and contracts as interest in the application strengthens or weakens. The power of the open-source team is derived from constant peer review and design/code refactoring that results in a slow progression toward an optimal solution.

15.4 TECHNOLOGY DIRECTIONS

Software engineering is about far more than technology—it's about people and their ability to communicate their needs and innovate to make those needs a reality. Whenever people are involved, change occurs slowly in fits and starts. It's only when a "tipping point" is reached, that a technology cascades across the software engineering community and broad-based change truly does occur.

A few trends in process, methods, and tools that are likely to have some influence on software engineering over the next decade are listed below.

15.4.1 PROCESS TRENDS

Possible process trends over the next decade are

a) **SPI frameworks** - will emphasize "strategies that focus on goal orientation and product innovation." A stable, step-by-step road map for SPI may have to be replaced with a framework that emphasizes short-term goals that have a product orientation. If the requirements for a new software-based product line will emerge over a series of incremental product releases (to be delivered to end users via the Web) the software organization may recognize the need to improve its ability to manage change. Process improvements associated with change management must be coordinated with the release cycle of the product in a way that will improve change management while at the same time not being disruptive.

b) **Process changes** will be driven by the needs of practitioners and should start from the bottom up. By focusing SPI efforts narrowly and working from the bottom up, practitioners will begin to see substantive changes early. Change that makes real difference in the way that, the software engineering works is conducted.

c) Greater emphasis will be placed on the return-on investment of SPI activities

$$\text{ROI} = \frac{\Sigma(\textit{benefits}) - \Sigma(\textit{costs})}{\Sigma(\textit{costs})} \times 100\%$$

d) Expertise in sociology and anthropology may have as much or more to do with successful SPI as other, more technical disciplines. SPI changes organizational culture and cultural change involves individuals and groups of people. Much can be learned by examining the sociology of groups to better understand effective ways to introduce change.

e) New modes of learning may facilitate the transition to a more effective software process. In this context, "learning" implies learning from successes and mistakes. A software organization that collects metrics allows itself to understand how elements of a process affect the quality of the end product.

f) Automated software process technology (SPT) will move away from global process management (broad based support of the entire software process) to focus on those aspects of the software process that can best benefit from automation.

15.4.2 THE GRAND CHALLENGE

There is one trend that is undeniable software-based systems will undoubtedly become bigger and more complex as time passes. It is the engineering of these large, complex systems, regardless of delivery platform or application domain, the poses the "grand challenge" for software engineers. New approaches are created to understanding system models and using those models as a basis for the construction of high-quality next generation software.

As the software engineering community develops new model-driven approaches to the representation of system requirements and design, the following characteristics must be addressed:

a) **Multi-functionality**—as digital devices evolve into their second and third generation, they have begun to deliver a rich set of sometimes unrelated functions. The mobile phone, once considered a communication device, is now used for taking photos, keeping a calendar, navigating a journey, and as a music player. If open-world interfaces come to pass, these mobile devices will be used for much more over the coming years.

b) **Reactivity and timeliness**—digital devices increasingly interact with the real world and must react to external stimuli in a timely manner. They must interface with a broad array of sensors and must respond in a time frame that is appropriate to the task at hand. New methods must be developed that

   1. Help software engineers predict the timing of various reactive features.
   2. Implement those features in a way that makes the feature less machine dependent and more portable.

c) **New modes of user interaction**—the keyboard and mouse work well in a PC environment, but open-world trends for software mean that new modes of interaction must be modeled and implemented. Whether these new approaches use multitouch interfaces, voice recognition, or direct mind interfaces, new generations of software for digital devices must model these new human-computer interfaces.

d) **Complex architectures**—a luxury automobile has over 2000 functions controlled by software that resides within a complex hardware architecture that includes multiple CPUs, a sophisticated bus structure, actuators, sensors, an increasingly sophisticated human interface, and many safety-rated components. Even more complex systems are on the immediate horizon, presenting significant challenges for software designers.

e) **Heterogeneous, distributed systems**—the real-time components of any modern embedded system can be connected via an internal bus, a wireless network, or across the Internet (or all three).

f) **Criticality**—software has become the pivotal component in virtually all business-critical systems and in most safety-critical systems. Yet, the software engineering community has only begun to apply even the most basic principles of software safety.

g) **Maintenance variability**—the life of software within a digital device rarely lasts beyond 3 to 5 years, but the complex avionics systems within an aircraft has a useful life of at least 20 years. Software characteristics can be managed only if the software engineering community develops

1. A more effective distributed and collaborative software engineering philosophy
2. Better requirements engineering approaches
3. A more robust approach to model-driven development
4. Better software tools.

15.4.3 COLLABORATIVE DEVELOPMENT

Today, software engineers collaborate across time zones and international boundaries, and every one of them must share information. The challenge over the next decade is to develop methods and tools. They identify a number of success factors that lead to successful collaborative efforts:

a. **Shared goals** —project goals must be clearly enunciated, and all stakeholders must understand them and agree with their intent.

b. **Shared culture** —cultural differences should be clearly defined, and an educational approach (that will help to mitigate those differences) and a communication approach (that will facilitate knowledge transfer) should be developed.

c. **Shared process** —in some ways, process serves as the skeleton of a collaborative project, providing a uniform means for assessing progress and direction and introducing a common technical “language” for all team members.

d. **Shared responsibility**—every team member must recognize the importance of requirements engineering and work to provide the best possible definition of the system.

15.4.4 REQUIREMENT ENGINEERING

The success or failure of these actions has a very strong influence on the success or failure of the entire software engineering process. Software requirements have a tendency to keep changing, and with the advent of open-world systems, emergent requirements (and near-continuous change) may become the norm.

Today, most “informal” requirements engineering approaches begin with the creation of user scenarios (e.g., use cases). More formal approaches create one or more requirements models and use these as a basis for design. Formal methods enable a software engineer to represent requirements using a verifiable mathematical notation. All can work reasonably well when requirements are stable, but do not readily solve the problem of dynamic or emergent requirements.

There are a number of distinct requirements engineering research directions. They are

- Natural language processing from translated textual descriptions into more structured representations (e.g., analysis classes)
- Greater reliance on databases for structuring and understanding software requirements
- The use of RE patterns to describe typical problems and solutions when requirements engineering tasks are conducted
- Goal-oriented requirements engineering.

To improve the manner in which requirements are defined, the software engineering community will likely implement three distinct sub processes as RE is conducted:

(1) Improved knowledge acquisition and knowledge sharing that allows more complete understanding of application domain constraints and stakeholder needs,
(2) Greater emphasis on iteration as requirements are defined, and
(3) More effective communication and coordination tools that enable all stakeholders to collaborate effectively

The RE sub processes will only succeed if they are properly integrated into an evolving approach to software engineering. As pattern-based problem solving and component-based solutions begin to dominate many application domains, RE must accommodate the desire for agility and the inherent emergent requirements that result. The concurrent nature of many

software engineering process models means that RE will be integrated with design and construction activities.

As a consequence, the notion of a static "software specification" is beginning to disappear, to be replaced by "value driven requirements" derived as stakeholders respond to features and functions delivered in early software increments.

15.4.5 MODEL-DRIVEN SOFTWARE DEVELOPMENT

Model-driven software development couples domain-specific modeling languages with transformation engines and generators in a way that facilitates the representation of abstraction at high levels and then transforms it into lower levels

Domain-specific modeling languages (DSMLs) represent "application structure, behavior and requirements within particular application domains" described with meta-models that "define the relationships among concepts in the domain and precisely specify the key semantics and constraints associated with these domain concepts.

15.4.6 POSTMODERN DESIGN

As the design of software components moves closer to algorithmic detail, a designer begins to represent the component at a level of abstraction that is close to code. Postmodern design will continue to emphasize the importance of software architecture. A designer must state an architectural issue, make a decision that addresses the issue, and then clearly define the assumptions, constraints, and implications that the decision places on the software as a whole. Aspect-oriented software development or model-driven software development may become important design approaches in the years ahead.

15.4.7 TEST-DRIVEN DEVELOPMENT

Requirements for a software component serve as the basis for the creation of a series of test cases that exercise the interface and attempt to find errors in the data structures and functionality delivered by the component. TDD is not really a new technology but rather a trend that emphasizes the design of test cases before the creation of source code.

The TDD process follows the simple procedural flow illustrated in Fig 15 4.7.1

- Before the first small segment of code is created, a software engineer creates a test to exercise the code (to try to make the code fail).
- The code is then written to satisfy the test.

- If it passes, a new test is created for the next segment of code to be developed.
- The process continues until the component is fully coded and all tests execute without error.
- If any test succeeds in finding an error, the existing code is refactored (corrected) and all tests created to that point are re-executed.
- This iterative flow continues until there are no tests left to be created, implying that the component meets all requirements defined for it.

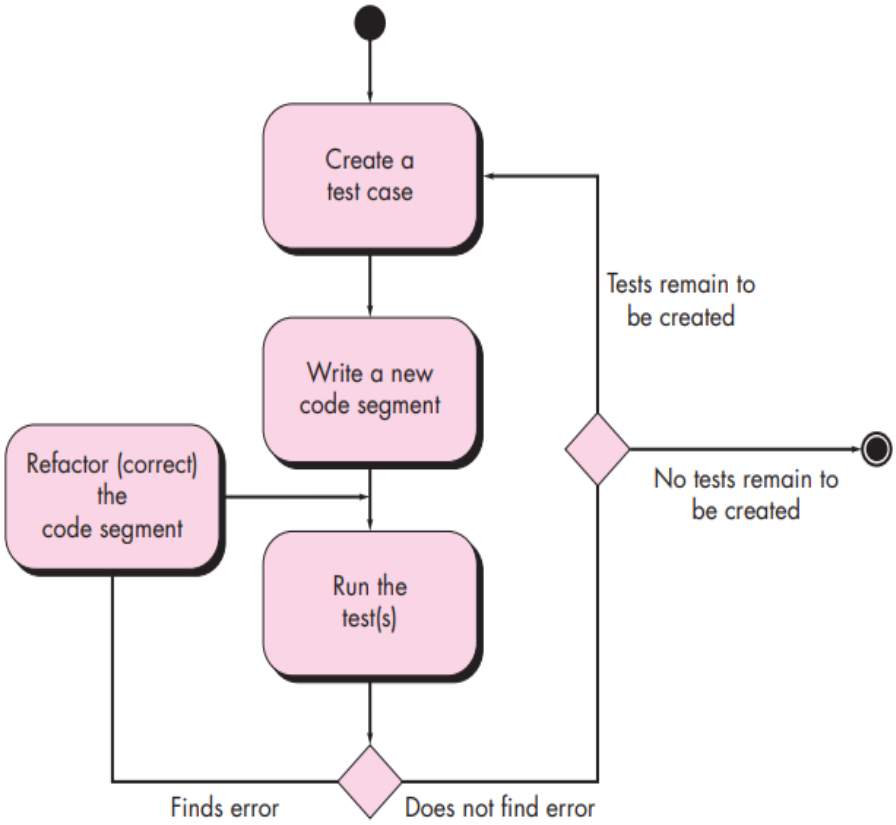


Fig 15. 4.7.1 Test-driven development process flow

During TDD, code is developed in very small increments (one sub function at a time), and no code is written until a test exists to exercise it. Each iteration results in one or more new tests that are added to a regression test suite that is run with every change. This is done to ensure that the new code has not generated side effects that cause errors in the older code.

In TDD, tests drive the detailed component design and the resultant source code. The results of these tests cause immediate modifications to the component design (via the code), and more important, the resultant component (when completed) has been verified in stand-alone fashion.

15.5 TOOLS RELATED TRENDS

Hundreds of industry-grade software engineering tools are introduced each year.

- The majority is provided by tools vendors who claim that their tool will improve project management, or requirements analysis, or design modeling, or code generation, or testing, or change management, or any of the many software engineering activities, actions, and tasks.
- Other tools have been developed as open-source offerings. The majority of open-source tools focus on "programming" activities with a specific emphasis on the construction activity (particularly code generation).
- Other tools grow out of research efforts at universities and government labs.
- At the industry level, the most comprehensive tools packages form software engineering environments that integrate a collection of individual tools around a central database (repository).

When considered as a whole, an SEE integrates information across the software process and assists in the collaboration that is required for many large, complex software-based systems. There is also a substantial time lag between the introduction of new technology solutions (e.g., model-driven software development) and the availability of viable SEEs that support the new technology.

15.5.1 TOOLS THAT RESPOND TO SOFT TRENDS

The soft trends need to manage complexity, accommodate emergent requirements, establish process models that embrace change, coordinate global teams with a changing talent mix, among others—suggest a new era in which tools support for stakeholder collaboration will become as important as tools support for technology.

A collaborative SEE "supports co-operation and communication among software engineers belonging to distributed development teams involved in modeling, controlling, and measuring software development and maintenance processes.

One example of research in this area is GENESIS—a generalized, open-source environment designed to support collaborative software engineering work. Figure 15.5.1 illustrates architecture for a collaborative SEE.

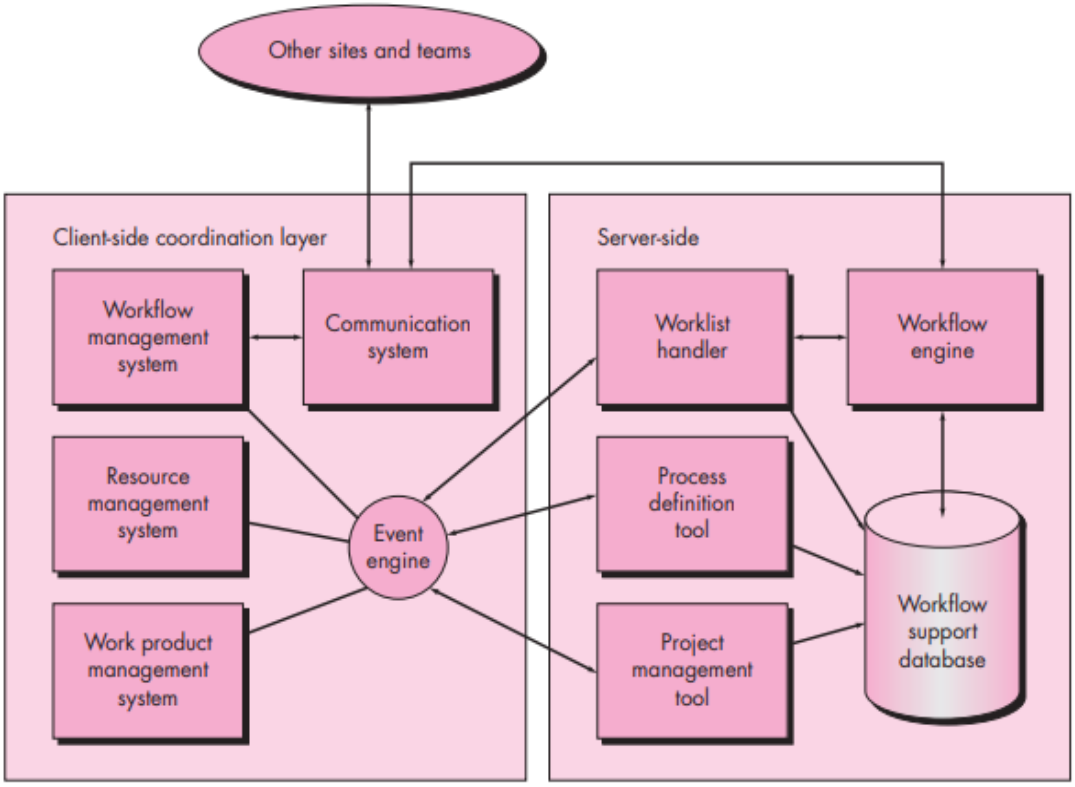


Fig 15.5.1 Collaborative SEE architecture

The architecture, based on the GENESIS environment, is constructed of subsystems that are integrated within a common Web client and is complemented by server-based components that provide support for all clients. Each development organization has its own client-side subsystems that communicate to other clients. The functions of the client-side components are as follows

- **A resource management subsystem** manages the allocation of human resources to different projects or subprojects;
- **A work product management system** is "responsible for the creation, modification, deletion," indexing, searching, and storage of all software engineering work products.
- **A workflow management subsystem** coordinates the definition, instantiation, and implementation of software process activities, actions, and tasks;
- **An event engine** "collects events" that occur during the software process (e.g., a successful review of a work product, the completion of unit testing of a component) and notifies others;
- **A communication system** supports both synchronous and asynchronous communication among the distributed teams.

On the server side, four components share a workflow support database. The components implement the following functions:

- **Process definition**—a tool set that enables a team to define new process activities, actions, or tasks and defines the rules that govern how these process elements interact with one another and the work products they produce.
- **Project management**—a tool set that allows the team to build a project plan and coordinate the plan with other teams or projects.
- **Workflow engine**—"interacts with the event engine to propagate events that are relevant for the execution of cooperating processes executed on other sites".
- **Worklist handler**—interacts with the server-side database to provide a software engineer with information about the task currently under way or any future task that is derived from work that is currently being performed.

15.5.2 TOOLS THAT ADDRESS TECHNOLOGY TRENDS

One of the dominant trends in technology tools is the creation of a tool set that supports model-driven development with an emphasis on architecture-driven design. A new generation of tools will work in conjunction with a repository to

- Create models at all necessary levels of abstraction,
- Establish relationships between the various models,
- Translate models at one level of abstraction to another level (e.g., translate a design model into source code),
- Manage changes and versions,
- Coordinate quality control and assurance actions against the software models.

In addition to complete software engineering environments, point-solution tools that address everything from requirements gathering to design/code refactoring to testing will continue to evolve and become more functionally capable. In some instances, modeling and testing tools targeted at a specific application domain will provide enhanced benefit when compared to their generic equivalents.

15.6 AGILE PROJECT MANAGEMENT

Agile project management, a major trend is an **iterative** approach to delivering a project throughout its life cycle. Iterative or agile life cycles are composed of

several **iterations** or **incremental** steps towards the completion of a project. Iterative approaches are frequently used in **software development** projects to promote velocity and adaptability since the benefit of iteration is that you can adjust as you go along rather than following a linear path.

One of the aims of an agile or iterative approach is to release benefits throughout the process rather than only at the end. At the core, agile projects should exhibit central values and behaviors of trust, flexibility, empowerment and collaboration.

Agile software development is considered

- In highly dynamic market environments where a strong focus on end user requirements is mandatory
- By most interviewees to be the weapon of choice when projects are explorative, are about innovative development, offer leeway for developers and do not involve too complex software architecture.

Regardless of the context in which agile software development is implemented, respondents clarified that agile software development must not serve as an empty label behind which developers hide chaotic software development practices. Agile software development neither frees developers from a priori specification of the system to be developed nor renders systematic planning and procedures obsolete. However, it still presupposes developers keep track of the global system to be developed.

Advantages

- **Speeding Up software development in Unstable Environments**: Generally, agile software development allows software development processes to speed up, account for the dynamics of emerging requirements, and integrate customers/users into the SD process.
- **Empowering Developers and Teams**: From the developers' point of view, agile software development strengthens entrepreneurial mentalities as well as heightens individual freedom and responsibility by equalizing and empowering team members.
- **Improving Coordination**: Agile software development decreases loss of time spent for meetings and coordination; it fosters transparency and collective learning, and improves predictability and productivity.

The effectiveness of agile methodologies depends on the way they are implemented by a given company; in this sense, experts stressed that introducing agility does entail a change in business culture and organizational structures of the wider organization indeed.

15.6.1 IT SECURITY IMPLICATIONS OF AGILE SOFTWARE DEVELOPMENT

From the analysis of expert's statements, it follows that these implications eventually revolve around the notion of expertise.

(a) Changing Requirements and Developers' Expertise - One of the fundamental agile values is to welcome changing requirements, even late in development. However, altering or introducing new requirements may have security implications. Also, change in requirements, design and a feature threatens to bring about problems for organizations in so far as any change on the software development level may have repercussions for the management, documentation and support level. Moreover, when outsourcing components, it may prove problematic to change requirements late in the process. The analogy to functional features says that they can't be modularized nor integrated in the final stages or a posteriori.

Agile software development demands

- A great deal of individual developers in terms of security expertise.
- SD teams tend to be more fluid, every team member needs to have security awareness and expertise.
- Every team member must be able to specify security relevant requirements as well as judge whether and how some change in requirements affects security.
- Thus, agile SD increases the security expertise that any individual developer is required to have.

b) Systematic Processes and Scalability of Expertise - In so far as agile SD emphasizes the individual and downplays processes, it may evoke a weakening of the overall system's perspective. This may prove problematic, since an overall system's view is necessary to take care of security. Agile SD challenges companies to implement systematic security processes, which allow security expertise to be distributed as widely as possible without overburdening individuals with security concerns. Hence any secure software development life cycle (SDLC) must be clearly defined and tailored specifically to the company adopting it.

In spite of the specifics of implementing an SDLC in relation to a company's organizational culture, experts mentioned three measures that will be considered:

(i) Security Early On: In agile SD it is even more crucial to integrate security into the SD process early on. Threat models and solutions are to be specified early on and followed up over the whole SDLC; there must be early quality assurance, early (immediate) and systematic testing instructed by experts; immediate feedback and fixing is necessary in the case of vulnerability occurrences.

ii) Scalability of Security Expertise: As it is not very easy to implement feedback mechanisms for immediate bug fixing in large-scale organizations, and as individual developers cannot be expected to have detailed knowledge of any testing tool (though such knowledge is required so as to not have too many false positives etc.), security specialists are supposed to manage testing tools centrally, with the developers able to choose correctly and use them if required so as to make security expertise scalable.

iii) Strengthening Exchange: Along with the individual developers to be equipped with the skills and tools to realize and fix security issues, security experts need to acquire in turn agile SD expertise so as to know how to integrate security into such processes. In this sense, there is not only a need to provide developers with security expertise, but also to provide security experts with agility expertise.

15.7 CHECK YOUR PROGRESS

1. Which are the trends leads to technological innovation?
2. Define hype cycle
3. ___ can guide the direction of research and the technology that is derived as a consequence of research.
4. What is the aim of SPI frameworks?
5. Write the formula for Return on Investment (ROI)
6. DSML stands for ___.
7. Agile project management is an ___ approach.
8. What is the aim of Agile project management?

Answers to check your progress:

1. Bussiness, organizational, market and cultural trends
2. The hype cycle is a graphical representation of the life cycle stages a technology goes through from conception to maturity and widespread adoption.

3. Soft trends
4. SPI frameworks will emphasize strategies that focus on goal orientation and product innovation.
5. $$ROI = \frac{\Sigma(benefits) - \Sigma(costs)}{\Sigma(costs)} \times 100\%$$
6. Domain-Specific Modeling Languages
7. Iterative approach
8. The aim of an agile project management is to release benefits throughout the process rather than only at the end.

15.8 SUMMARY

The trends that have an effect on software engineering technology often come from the business, organizational, market, and cultural arenas. These "soft trends" can guide the direction of research and the technology that is derived as a consequence of research.

As a new technology is introduced, it moves through a life cycle that does not always lead to widespread adoption, even though original expectations are high. The degree to which any software engineering technology gains widespread adoption is tied to its ability to address the problems posed by both soft and hard trends.

Soft trends—the growing need for connectivity and collaboration, global projects, knowledge transfer, the impact of emerging economies, and the influence of human culture itself, lead to a set of challenges that span managing complexity and emergent requirements to juggling an ever-changing talent mix among geographically dispersed software teams.

Hard trends—the ever-accelerating pace of technology change—flow out of soft trends and affect the structure of the software and scope of the process and the manner in which a process framework is characterized. Collaborative development, new forms of requirements engineering, model-based and test-driven development, and postmodern design will change the methods landscape. Tools environments will respond to a growing need for communication and collaboration and at the same time integrate domain-specific point solutions that may change the nature of current software engineering tasks.

15.9 KEYWORDS

- The "**hype cycle**": It presents a realistic view of short-term technology integration. The long-term trend, however, is exponential.
- **Open-world software:** It encompasses ambient intelligence, context aware apps, and pervasive computing
- **Collaboration**: It involves the timely dissemination of information and an effective process for communication and decision making.
- **Model-driven approaches**: It address a continuing challenge for all software developers—how to represent software at a higher level of abstraction than code.
- **TDD**: It is a trend that emphasizes the design of test cases before the creation of source code.

15.10 QUESTIONS FOR SELF STUDY

1. Why does open-world software present a challenge to conventional software engineering approaches?
2. What are "emergent requirements" and why do they present a challenge to software engineers?
3. What are soft and hard trends? What is the importance of soft trends?
4. Explain briefly Technology evolution using technology innovation life cycle.
5. Describe the Hype cycle
6. Define open source. List its advantages.
7. Discuss few trends in process and methods that are likely to influence on software engineering over the next decade.
8. Explain briefly the success factors that lead to successful collaboration efforts in collaborative development.
9. Give short notes on the Requirement engineering.
10. Describe model-driven software development in your own words. Do the same for test driven development.
11. Explain TDD
12. Discuss about tools related trends.
13. Give an account of IT security implications of Agile software development.

15.11 REFERENCES

1. Agile Software Development: Principles Patterns and Practices - Robert C. Martin – StuDocu.
2. Software Engineering, A Practitioner's Approach – 7th Edition, Roger S.Pressman.
3. Software Project Management in Practice – Pankaj Jalote.
4. Software Engineering: Pearson New International Edition – Ian Sommerville, 2013.

UNIT-16: PRACTICING WITH AN ONLINE PROJECT MANAGEMENT TOOL

STRUCTURE

16.0 OBJECTIVES

After studying this unit, you will be able to:

- Learnt how to manage projects
- Describe about how to create folders
- Know about handling project
- Students are able to plan the project
- Learns project scheduling and time estimation

16.1 INTRODUCTION

Monday.com is a customizable web and mobile work management platform, designed to help teams and organizations with operational efficiency by tracking projects and workflows, visualizing data, and team collaboration. It includes automation capabilities and supports integrations with other work apps. Monday.com continuously ranks highly for project management software. In addition to its support for a wide range of use cases, the software is

loaded with functionality. It comes with collaboration features, timeline views, calendar views, time tracking features, and dozens of integrations.

16.2 OVERVIEW

Monday.com is an open source project management tool. Monday.com is now all-in-one free project management tool and customer relationship management (CRM) software. Monday.com is very easy to use and almost self-explanatory. It is a user-friendly, visually pleasing tool that enables small and large businesses to collaborate together on one easy to use platform. In the recent update, Monday.com has come up with a range of innovative solutions for businesses to choose from.

16.3 GETTING STARTED WITH MANDAY.COM

So, let's get you started on how to integrate your business with Monday.com and manage your projects and customers, all in one place. Free plan is ideal for new business start-ups or freelancers who want to get started with project management or customer relationship management activities. Two users can have access to the free subscription plan. Monday.com has a plan for small teams of 3-10 people (which also offers a 14-day free trial to start with initially). If you have a team of larger than 2 people then we recommend you begin with the standard plan because it gives you access to more advanced features like automation that are incredibly useful for saving time. Once you have set up the account, you can add your team members simply by adding their email addresses. We have to create an account in the Monday.com to manage your work. Below screenshots show the steps to of create an account.

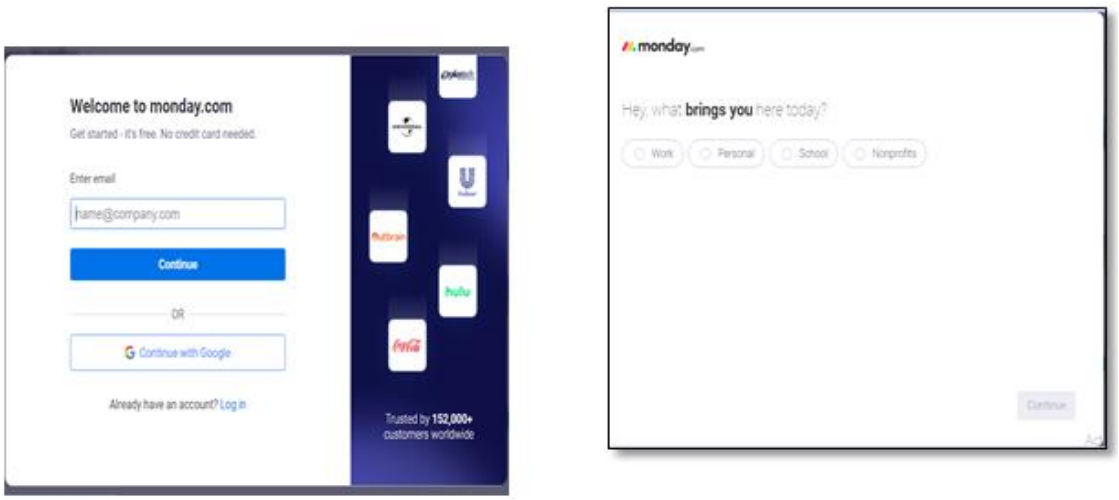


Figure 16.1 Welcome page to Monday.com

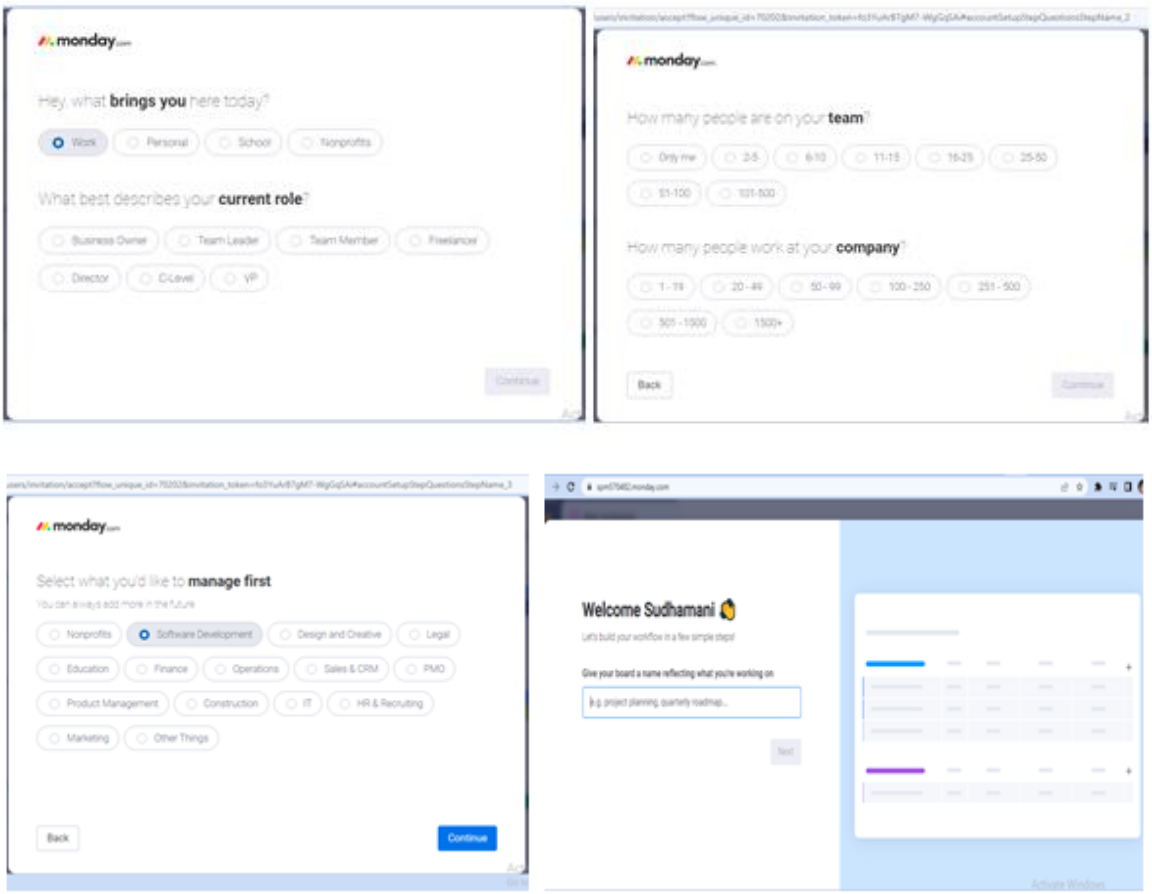


Figure 16.2. Screen shots to create an accoount

16.4 CREATE WORKSPACE

Workspaces are used by teams to organize and manage their accounts by departments, teams and projects. Workspaces can contain boards, dashboards, and folders. Any team member subscribed to a board from a closed workspace will have access to that board but will not be able to see anything else within the workspace. Main boards in closed workspaces are accessible only to team members who have joined the workspace.

The main center of Monday is the workspace where projects can be managed, set up campaigns, sales pipelines or CRM's. One workspace usually needed for beginners but Monday.com offers the facility to create more than one workspace. Monday.com has made it easy to organize this space. Here the figure shows the workspace creation.

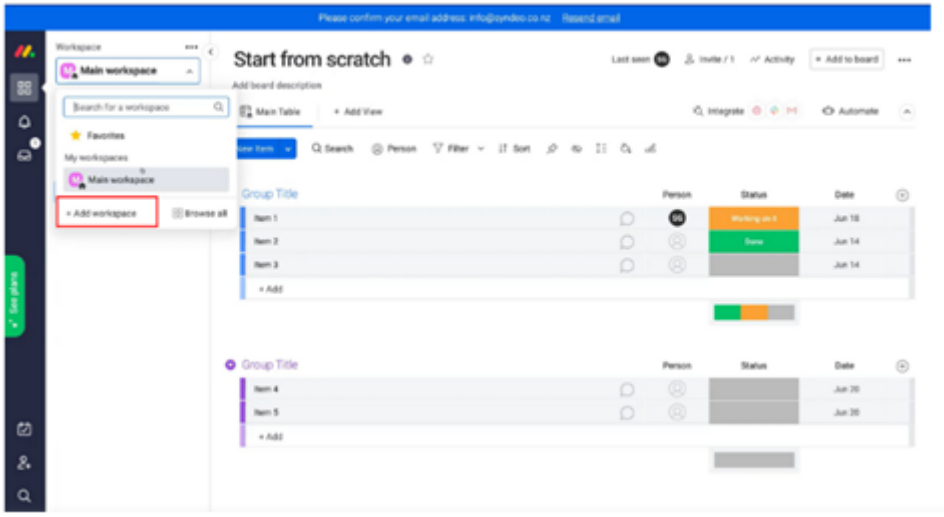


Figure 16.3 Workspace

16.5 CREATE FOLDERS

A folder is simply a storage space for boards. Folders make it easy to organize all of your boards so that your account can stay neat and clean. Workspaces are here to help your organization better manage multiple departments, teams and projects in one unified place.

We need to create folders before to starts with our first project. Monday.com offers us space to create different folders within your Workspace. This facility helps the user to organize multiple projects for multiple clients.

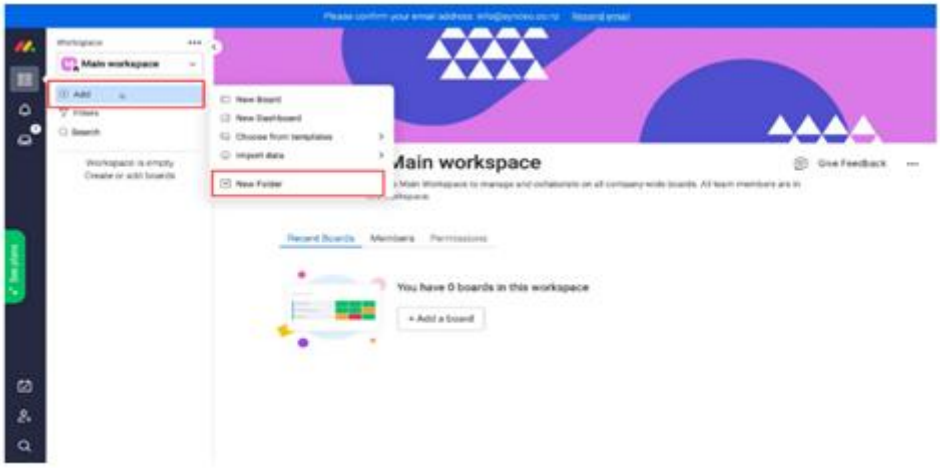


Figure 16.4 Folder creation

16.6 CREATE BOARDS

A board is where you can manage anything like project, board, roadmap, sales pipeline, and budget etc., the possibilities are endless. Main boards are visible to anyone who is a team member within your account whether they are viewers or members. The board is a view of a selection of issues that you can use to see and update them. It displays them in columns, with each column representing a step in your process for them. A dashboard is a place to collect together a set of reports that people might find useful.

We can create boards that are specific to the activity of a project, and Monday gives us the option to share the board with external users like clients. Clients can check the progress of their activities and monitor progress. Boards are treated as specific projects that are created. These projects can be managed in the folders in your Workspace.

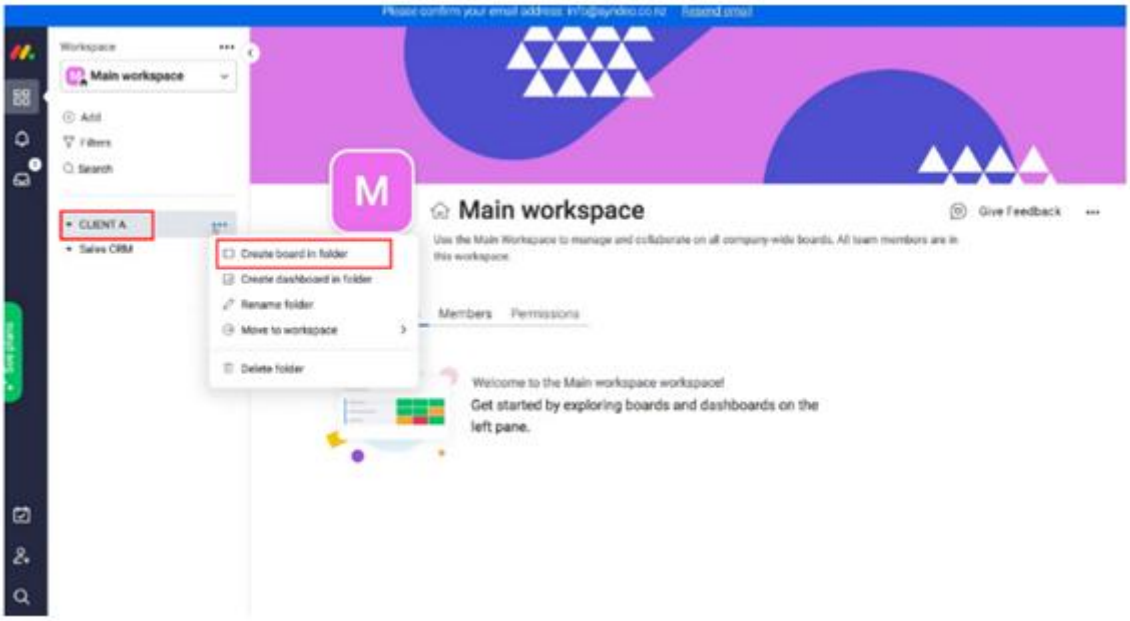


Figure 16.5 Board creation

So, new project can be created by click on the ‘create new board’ button on left pane then You can select from various templates for the kind of board (project) you wish to create, like Marketing, Content Production, Project Management or Sales and CRM-based projects.

SET UP A NEW MONDAY.COM BOARD

Once we have created an account, with 200+ customizable templates, it’s easy for us to choose find something that matches the business’s needs. Managing growth projects, sales teams, budgets, HR or a dev sprint, you can try the high level project plan template, the sales

process template, the budget tracker template, the employee onboarding template, or the sprint planning template.

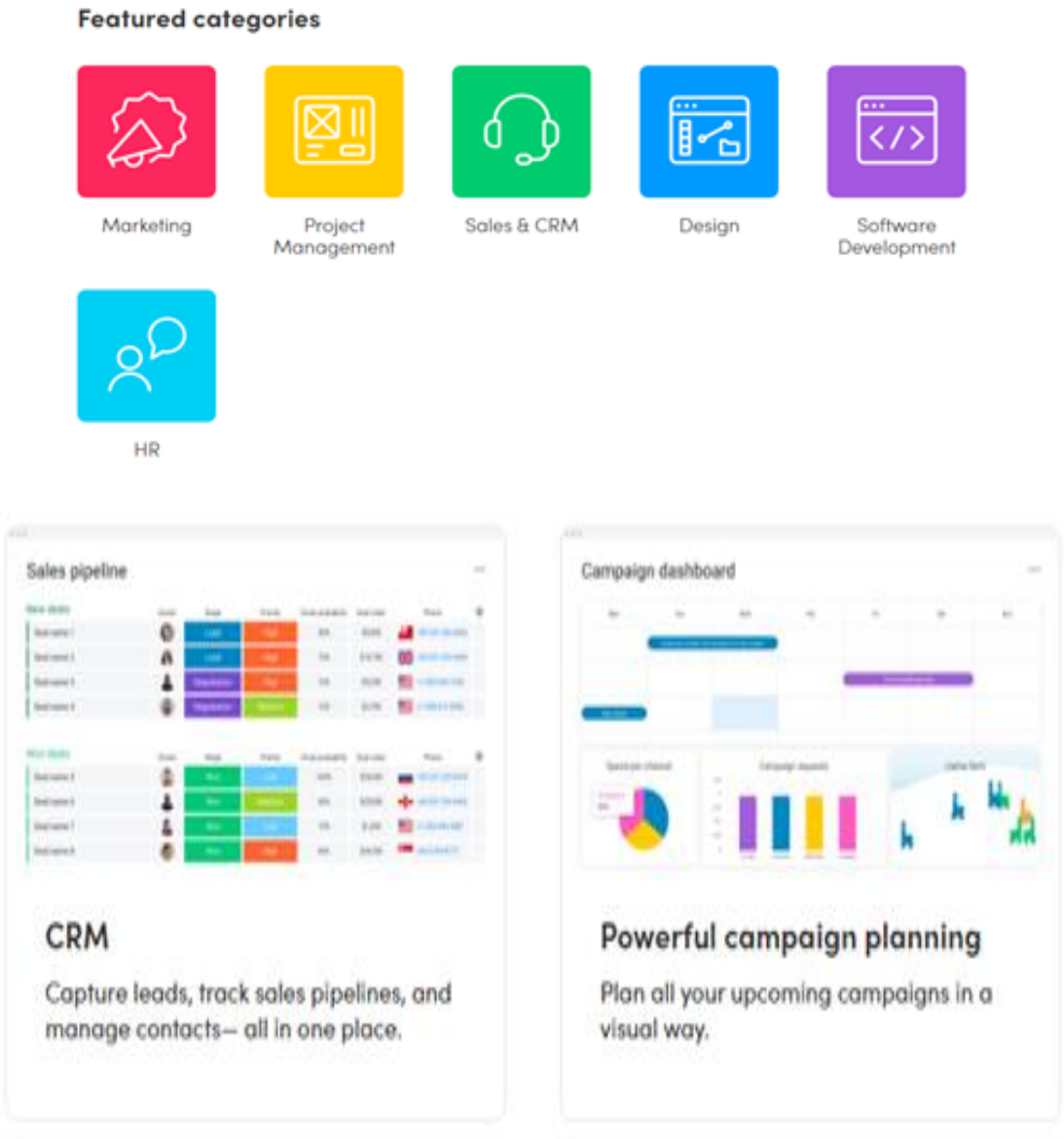


Figure 16.6. Categories and views in Monday.com

For instance, to lead company's marketing team, the editorial calendar and client management templates can be selected.

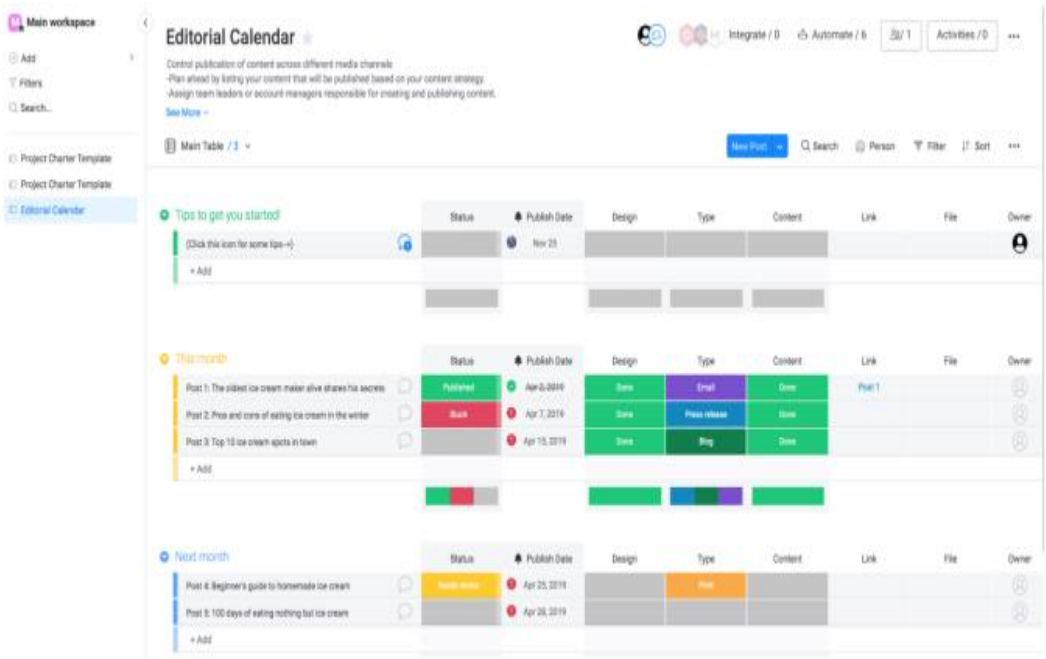


Figure 16.7. Editorial calender in Monday.com

BUILD YOUR MONDAY.COM DASHBOARD

Once you've got your monday.com boards and team in place (team creation is discussed in next section), it's time to set up your dashboards. monday.com dashboards provide you with a clear overview of your current projects and boards. You can customize dashboards by adding different widgets.

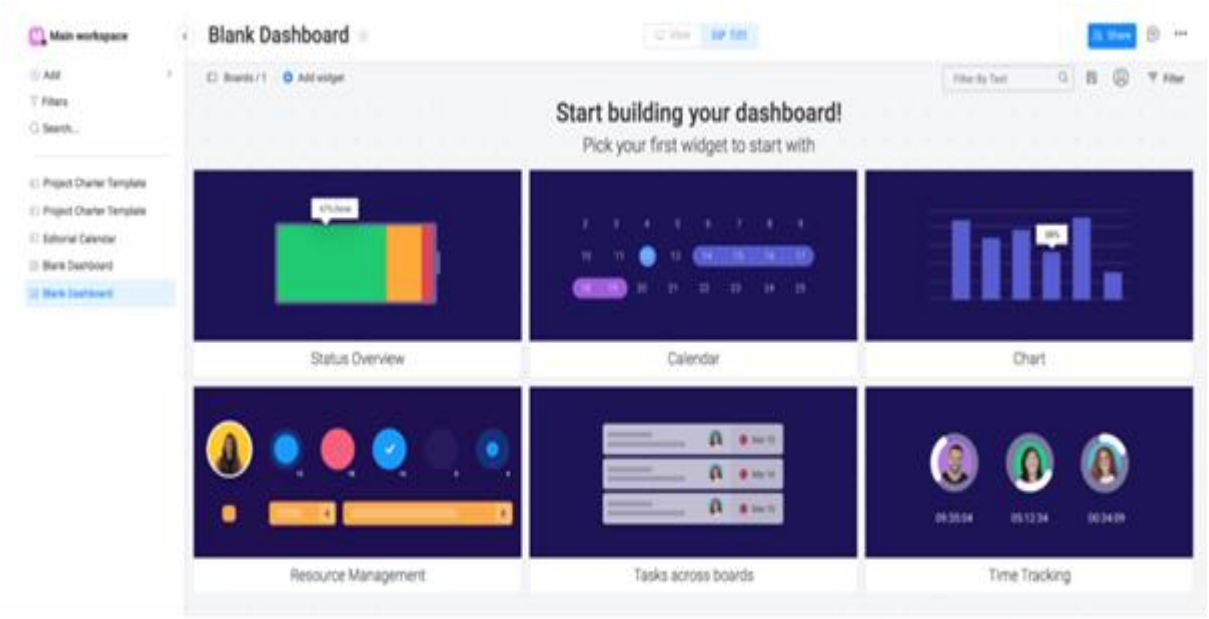


Figure 16.7. Dashboards in Monday.com

INCLUDING STATUSES AND DATES

Hitting deadlines and keeping track of all the different moving parts is a fundamental part of effective project management. Adding a start date and a finish time helps map out your project's overall timeline. You can include dates by adding a date column. Equally, a status column can give you a quick indication of how each task is going. Once you've added a status column, users can update whether a task is in progress, complete, or if they're stuck.

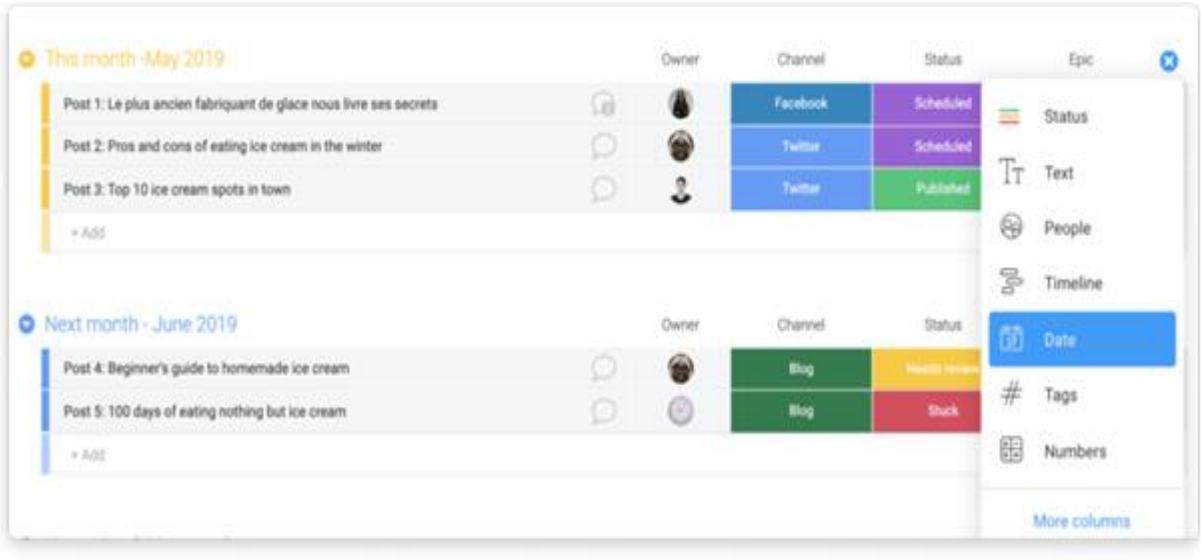


Figure 16.7. Monthly view in Monday.com

DISPLAY VIEWS

Depending on the business and project requirements, you may have a preferred way of displaying your timetables, Workflows, and processes. Here are the available views in Monday.

- **Kanban** is great for helping teams collaborate by displaying all project-related Workflows and sprints. Intuitive color-coding makes it easy to see at what stage of the project each task is at. Users who prefers brightly colored may choose Kanban board that maps out each section of the project.

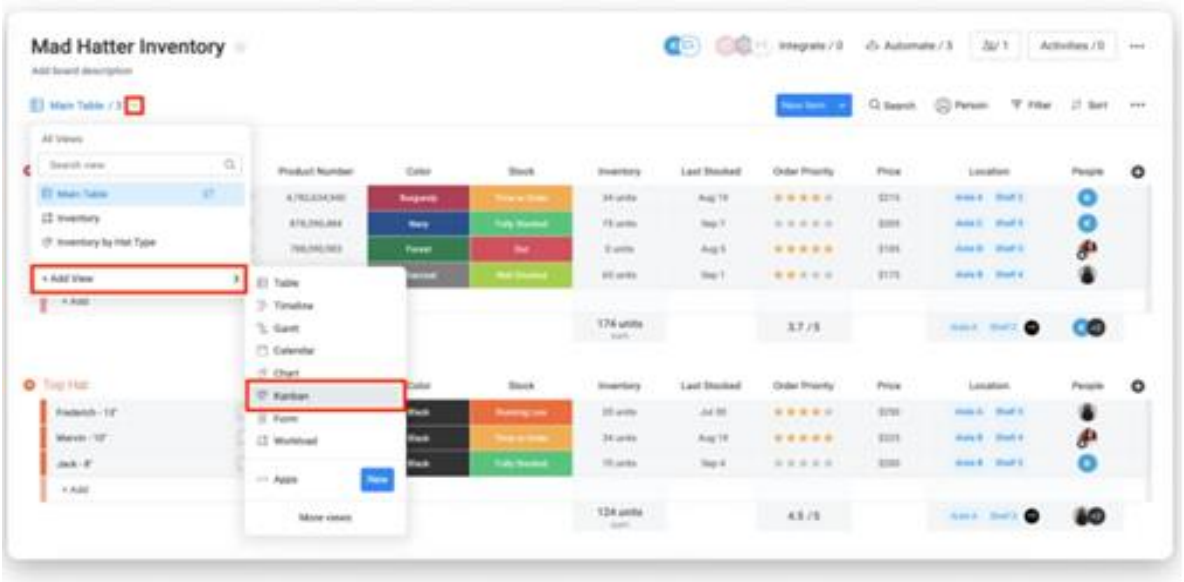


Figure 16.8. Display sample in Monday.com

- **Calendar** view is effective for planning out your team's monthly, quarterly, or yearly workload.

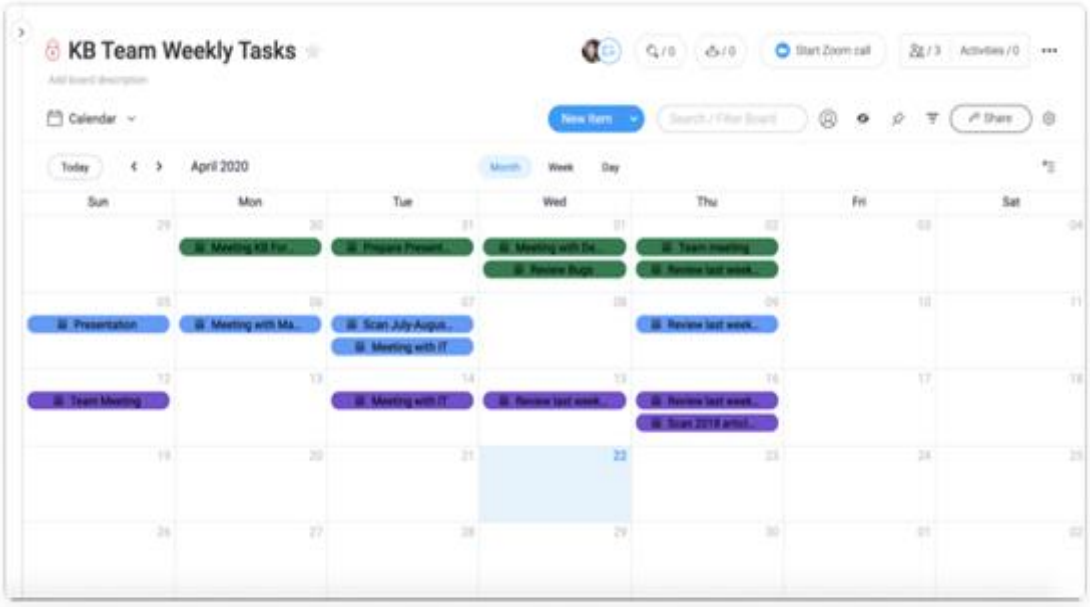


Figure 16.9. Task view screen

- **Gantt** view is best for mapping out project deadlines and checking in on task timelines in the form of a Gantt chart. A Gantt chart is a horizontal bar chart that shows a project's planned schedule the Gantt chart represents a task, while the dates are laid out horizontally and its tasks or events between a start and finish date.

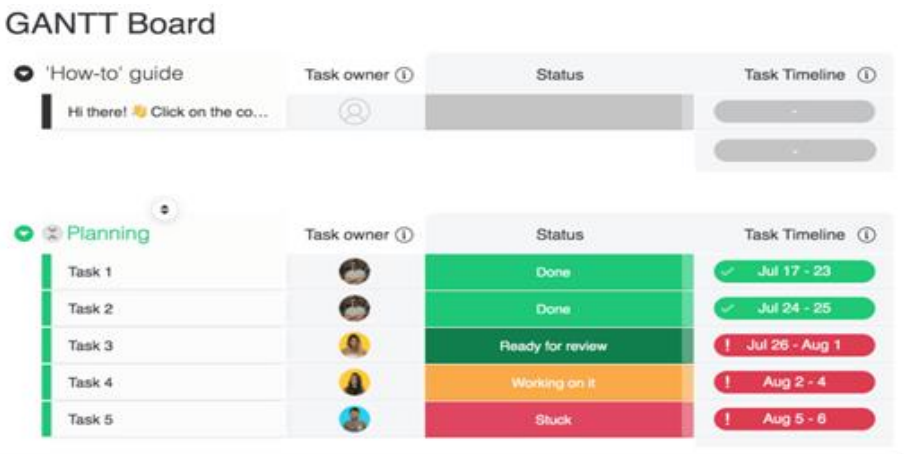


Figure 16.10. Gantt chart in Monday.com

- **Chart** helps you review all your project-related events and tasks.

Figure 16.11. chart view in Monday.com

- **Timeline** is effective for giving you a full overview of who is working on which projects when.

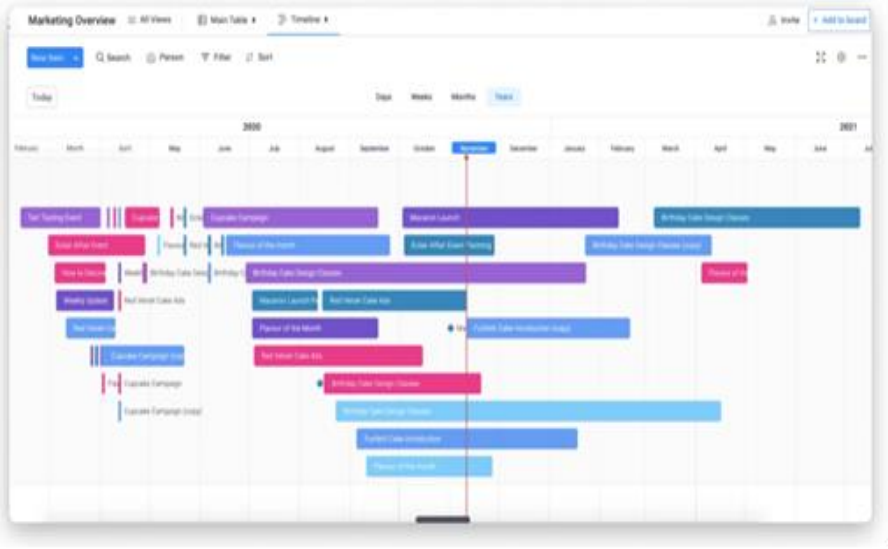


Figure 16.12. Timeline view in Monday.com

16.7 TEAM

Open communication and team collaboration is the key to successful project management. A team is any group of individuals who are working together to achieve a shared goal. In the workplace, team can be a department, a project team, a functional department team, a cross-functional or cross-departmental team, management teams, or whatever other type of team works in the company. The individuals in the teams on monday.com account, can be notified, updated, and communicated with all together as a team.

We need to click on your profile picture and select teams. Here, you can see all the existing team members on your account and add new team members. Once you've selected team members and named your team, we recommend choosing a team photo, which will make it much easier to identify who's working on what. Creating **teams** allows you to easily notify, update, assign tasks, and chat with teammates throughout your account all in one go.

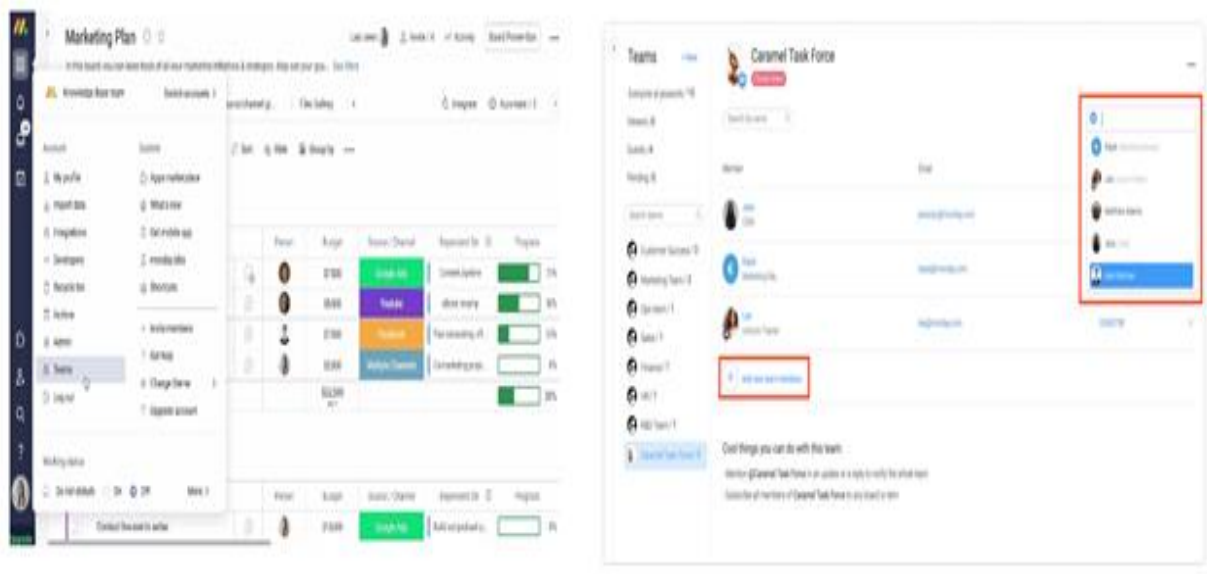


Figure 16.12. Team members visibilty page in Monday.com

Furthermore, **teams** can provide you with another way to keep information secure on your account. By sharing specific Private or Shareable boards with a team, you can ensure that the right people have access to the right information at all times.

16.8 CREATING GROUPS

A group is a color-coded section on your board that contains the items in rows and helps to categorize and organize them in any way that we wish. When creating a board, you then need to create your groups. A group can represent anything from a week, a month, a specific step of a project, client, or whatever you want.

We can create multiple groups to divide the tasks that you have to perform for the client, like client requirements, testing, in progress and completed tasks.

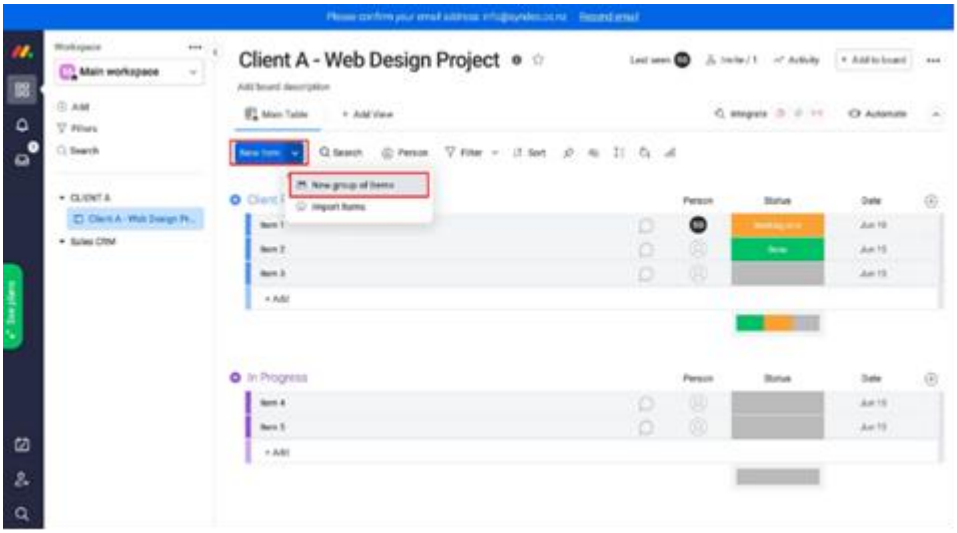


Figure 16.13. Groups of items in Monday.com

Different groups contain tasks, campaigns or features related to one project. They can all be organized according to the progress of the project. Each group can be named according to its completion of tasks. Then you can easily drag the tasks from one phase to another.

16.9 CREATING ITEMS

An item is an individual row or line item in a group, there are a few different ways that you can add a new item to your board. Items are tasks that need to complete the project group. Items are the deliverables and Monday project management software makes it a simple process to manage them.

We can simply click the ‘add item’ button and add the deliverables. Once the task is completed, we can move the item from one group to another.

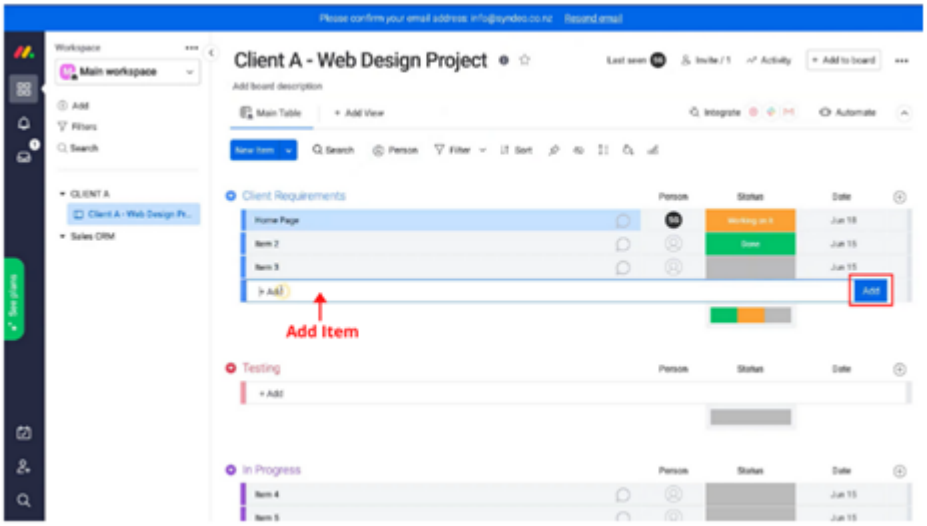


Figure 16.14. Adding items to the groups in Monday.com

CREATE COLUMNS

Within items, just beside each item, Monday.com software offers a feature to add columns to define various things like its status, due date, and name of the employee working on it, and more. One of the default columns is called people. We can define which person is working on the task. There is also an option to add more people to the task by simply adding their names to the people's column via email address.

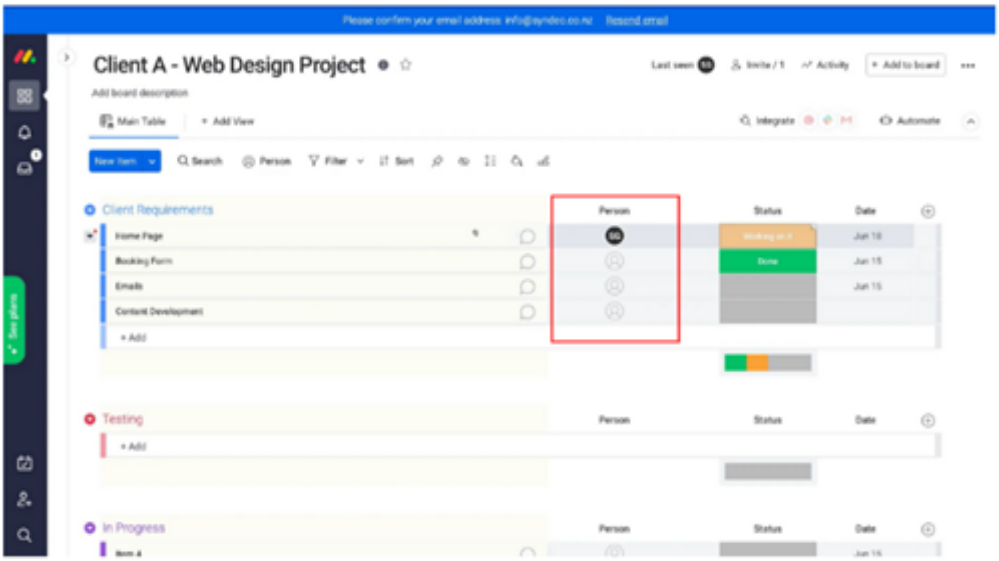


Figure 16.15. Adding columns in Monday.com

A simple right-click on the column button will open a drop-down menu of various items for which you can add a column. They can be things like timeline, tags, dates, sub-items, formula or text, and much more.

16.10 AUTOMATE

Manual work can be a huge drag on the team's overall productivity and set the project back. Monday.com Automations help relieve some of this tedious work. The best part is that the platform handles the technical side and this does not need to do any coding. Automations allow you to set any event to trigger an action. For instance, when a team member completes a task, you could set it to trigger a notification to the team leader.

Hence, automate allows us to setup pre-decided automations within the projects. We can observe a status here saying "when **Status** changes to **something**, notify **someone**". So here, we can define the parameters for status, something and someone. You can decide whom to notify when the status of an item or task changes from one status to another.

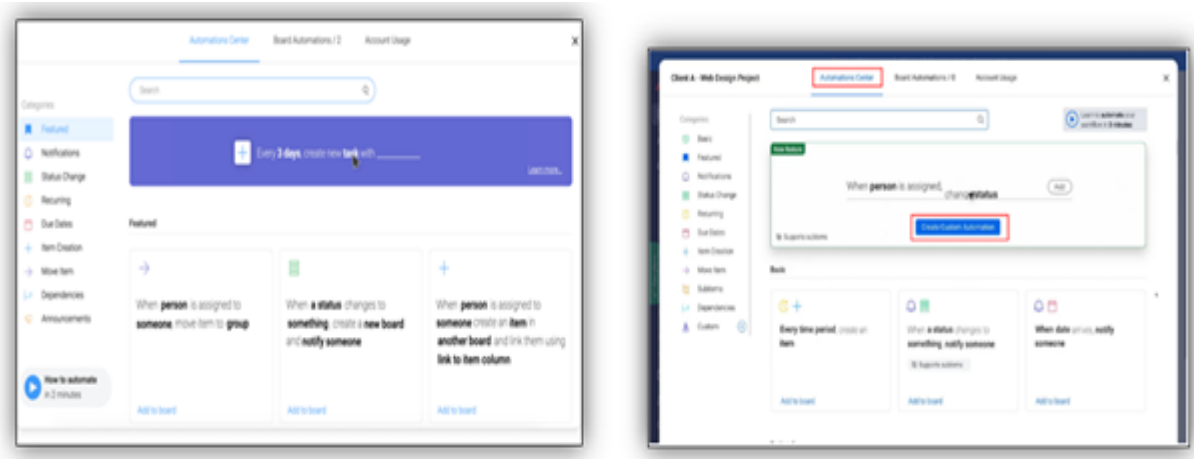


Figure 16.15. Project automation

For example, you can define the software to notify '**John**' when '**home page development**' task's status changes from '**in-progress**' to '**complete**'. This is just one example. There is an abundance of automations that you can leverage to reduce many of the manual, messy and time-consuming activities.

16.11 CHECK YOUR PROGRESS

1. Monday.com is a customizable web and mobile work management platform. State True/False.
2. Workspaces are used by teams to organize and manage their _____by departments, teams and projects.
3. A folder is simply a storage space for boards. State True/False.

4. Calendar view is effective for planning out your team's monthly, quarterly, or yearly_____.
5. Write the full form of (CRM) software.

Answer to check your progress:

1. True
2. Accounts
3. True
4. workload
5. customer relationship management

16.12 SUMMARY

Monday.com is highly customizable templates and dashboards have the power to transform the way the business works. While many other services out there might describe themselves as project management tools, monday.com is a complete Work OS — a workspace that allows you to do so much more than assign a task to the graphic designer from choosing multiple views to building the dashboard and collaborating with teammates, there is no limit to what we create and manage with monday.com. If you are keen to get started, give the monday.com platform a go and put all of these tips into practice.

16.13 KEYWORDS

- **Folder:** a folder is a *storage space* for board space.
- **Workspace:** monday.com Workspaces are used by teams to organize and manage their accounts by projects. Workspaces can contain boards, dashboards, and folders.
- **Boards:** A board is where you can manage anything like project.
- **CRM:** Customer relationship management.
- **FreeLancer:** Freelancing is a type of self-employment. Instead of being employed by a company, freelancers tend to work as self-employed, delivering their services on a contract or project basis,
- **Dashboards:** Dashboards are a great way to display what's important in just one place.

16.14 QUESTIONS FOR SELF STUDY

1. What is a folder? How can you create a folder in Monday.com?
2. What is workspace? Discuss the importance of workspace in project management.
3. What are boards and explain the necessary of boards?
4. What are the steps involved to create boards.
5. Discuss the different widgets of dashboards.
6. Explain various display views available in Monday.com.
7. Why team is necessary in project management? Discuss in detail how can you create a team and view team members.
8. Mention the steps to create multiple groups.
9. Discuss the steps to create items and how to view them.
10. Explain the importance of software automation.

16.15 REFERENCES

Monday.com