UNIT 2 SOFTWARE PROCESS MANAGEMENT

Structure **Page Nos.**

2.0 INTRODUCTION

This unit deals with the most important capital in any organisation which is Human Capital. The "people factor" is so important that the Software Engineering has developed a capability model to enhance the ability of software development organisations to undertake complex applications. Major Technology Fortune 500 companies contribute their success to their people. Even the profitability of a technology company falls, if certain key people leave the organisation. In this unit, we will discuss about Software Process Management, Human Resource Management, Software Teams, Problem Handling Guidelines and Managing the Software crisis.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- explain Software Process Management;
- know different Software team structures;
- know about Software crisis, and
- know the role of a Systems Analyst.

2.2 SOFTWARE PROCESS MANAGEMENT

Most design methodologies focus on designing a software from scratch. No design methodology really handles designing from an existing design, or changing a design to include new specifications. Change is an inherent property of software and is its strength. By having these methodologies, and people following them, the view is further strengthened that software should be developed as a “new product”. This clearly is inconsistent with the reality of things, and can be argued to hinder productivity growth, as developing a fresh product is likely to be more expensive than developing by reusing existing designs and implementations.

The separation of software process and software product is desirable and has helped software engineers understand them separately and give enough importance to the process also, which was neglected earlier. However, current process models and their implementations emphasize on process with the belief that software process determines most properties of software.

Some of the undesirable consequences that we have mentioned are the following:

(a) Tendency to make process document heavy and consequently not liked by people in the process, (b) neglect some of the important phases of development of software products, like software design, (c) inability of process to accept change, which is a fundamental property and strength of software, and (d) tendency to develop “processes” for creative activities like design.

However, some future trends in software engineering seem to be promising. There is an effort to build newer models that are more consistent with the properties of software. There is also an effort to develop more formal methods for software specification, design and verification, and software reuse is being targeted as the future software engineering technology – a target that will force development of new methods for design, which use existing designs.

2.3 HUMAN RESOURCE MANAGEMENT

Effective software project management focuses on the three Ps – people, problem, and process. The order is not arbitrary. The manager who forgets that software engineering work is an intensely human endeavour will never have success in project management. A manager who fails to encourage comprehensive customer communication early in the evolution of a project risks building an elegant solution for a different problem. Finally, the manager who pays little attention to the process runs the risk of inserting competent technical methods and tools into a vacuum.

The “people factor” is so important that the Software Engineering Science has developed a people management capability maturity model (PM-CMM) “to enhance the readiness of software organisations 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 capability maturity model defines the following key practice areas for software people: selection, performance management, training, compensation, career development, organisation and work design, and team/culture development. Organisations that achieve higher levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

Before a project can be planned, its objectives and scope should be established, alternative solution should be considered, and also technical as well as management constraints should be identified. Without this information, it is impossible to define reasonable (and accurate) estimates of the cost; an effective assessment of risk; a realistic breakdown of project tasks, and a manageable project schedule that provides a meaningful indication of progress.

The software developer and customer must meet to define project objectives and scope. In many cases, this activity begins as part of the system engineering process and continues as the first step in software requirements analysis. Objectives identify the overall goals of the project without considering how these goals will be achieved. Scope identifies the primary data, functions, and behaviours that characterise the problem, and more importantly, attempts to bound these characteristics in a quantitative manner.

Once the project objectives and scope are understood, alternative solutions are considered. The alternatives enable managers and professionals to select a "best" approach, given the constraints imposed by delivery deadlines, budgetary restrictions, personnel availability, technical interfaces, and other factors.

2.3.1 Software Process

A software process provides the framework from which a comprehensive plan for software development can be established. A small number of framework activities are applicable to all software projects, regardless of their size or complexity. A number of different *task sets* – tasks, milestones, deliverables, and quality assurance points – enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities – such as software quality assurance, software configuration management, and measurement – overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

The following are the assertions of some people in IT industry about the contributions to a successful IT project:

P1: If I had to pick one thing out that is most important in our Organisation, I'd say it's not the tools that we use, it's the people.

P2: The most important ingredient that was successful on this project was having smart people…. very little else matters in my opinion. … The most important thing you do for a project is selecting the staff. The success of the software development organisations is very much associated with the ability to recruit good people.

P 3: The only role I have in management is to ensure that I have good people – and that I provide an environment in which they can be productive.

Indeed, this is a compelling testimonial on the importance of people in the software engineering process. And yet, most of us often take people for granted. Managers argue that people are primary, but their actions sometimes believe their words.

Players who can be categorised into one of the five constituencies populate the software process (and every software project).

1. Senior managers, who define the business issues that often have significant influence on the project.
2. Project (technical) managers, who must plan, motivate, organise and control the practitioner who do software development.
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.
5. End users, who interact with the software once it is released for use.

For the above players to be effective, the project team must be organised in a way that maximizes each person's skills, and abilities permeate every software project.

2.3.2 Team Leaders

Project management is a people intensive activity. They simply don't have the right mix of people skills.

What do we look for when we select someone to lead a software project?

Motivation: The ability to encourage technical people to produce to their best of abilities.

The ability to encourage people to create and feel creative when they work within bounds established for a particular software project.

Successful project leaders apply a problem solving management style. That is, 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 in the team know (by words and, far more important, by actions) that quality counts and that it should not be compromised.

2.3.3 Problem Solving

An effective software project manager can diagnose the technical and organisational issues that are most relevant, systematically structure a solution or properly motivate other professionals 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 care of the project. S/he must have the confidence to assume control and permit 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 action that controlled risk taking will not be punished.

2.3.4 Influence and Team Building

An effective project manager must be able to “read” people; s/he must be able to understand verbal and non-verbal signals and react to the needs of the people sending these signals. The manager must control high stress situations.

☞ Check Your Progress 1

1) Effective software project management focuses on the three P’s namely, ___________, __________ and ______________.

2) Task sets include ________, __________ and _________.

2.4 THE SOFTWARE TEAM

The organisation 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 x people working for y years.

1) x 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) x individuals are assigned to m different tasks so that informal "teams" are established; an ad hoc team leader may be appointed; coordination among team members is the responsibility of the project manager.

3) x Individuals are organised into *t* teams; each team is assigned one or more functional task; each team has a specific structure that is defined for all teams working on the project; coordination is done by both the team and project manager.

Although, it is possible to voice *pro* and *con* arguments for each of the above approaches, there is a growing body of evidence that indicates that a formal team organisation is most productive.

The "best" team structure depends on the management style of an organisation, the number of people who will populate the team and their skill levels, and the overall problem difficulty. The following are the three generic team organisations:

2.4.1 Democratic Decentralised (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.

2.4.2 Controlled Decentralised (CD)

This software engineering team has a defined leader who coordinates specific tasks and secondary leaders who have responsibility for subtasks. Problem solving remains a group activity, but the team leader apportions implementation of solutions among subgroups. Communication among subgroups and individuals is horizontal. Vertical communication along the control hierarchy also occurs.

2.4.3 Controlled Centralised (CC)

Top-level problem solving and internal team coordination are managed by a team leader. Communication between the leader and team members is vertical.

The following are 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 the team will stay together (team lifetime).
- The degree to which the problem can be modularised.
- 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.

The length of time the team will "live together" affects team morale. It has been found that Democratic Decentralised team structure results in high morale and job satisfaction and is therefore good for long lifetime teams.

The Democratic Decentralised team structure is best applied to problems with relatively low modularity because of the higher volume of communication that is needed. When high modularity is possible, the Controlled Centralised or Controlled Decentralised structure will work well.

Controlled Centralised and Controlled Decentralised teams have been found to produce fewer defects than Democratic Decentralised teams, but these data have much to do with the specific quality assurance activities that are applied by the team. Decentarlised teams require more time to complete a project than a centralised structure but at the same time are best when high sociability is required.

Four “organisational paradigms” for software engineering teams are given below:

1. A closed paradigm structures a team along a traditional hierarchy of authority (similar to a Controlled Centralised team). Such teams can work well when producing software that is quite similar to past efforts, but they will be less likely to be innovative when working within the closed paradigm.
2. The random paradigm structures a team, loosely and depends on individual initiative of the team members. When innovation or technological breakthrough is required, teams following the random paradigm will excel. But such teams may struggle when “orderly performance” is required.
3. The open paradigm attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm. Work is performed collaboratively with heavy communication and consensus-based decision-making. Open paradigm team structures are well suited to the solution of a complex problem, but may not perform as efficiently as other teams.
4. The synchronous paradigm relies on the natural compartmentalisation of a problem with little active communication among themselves.

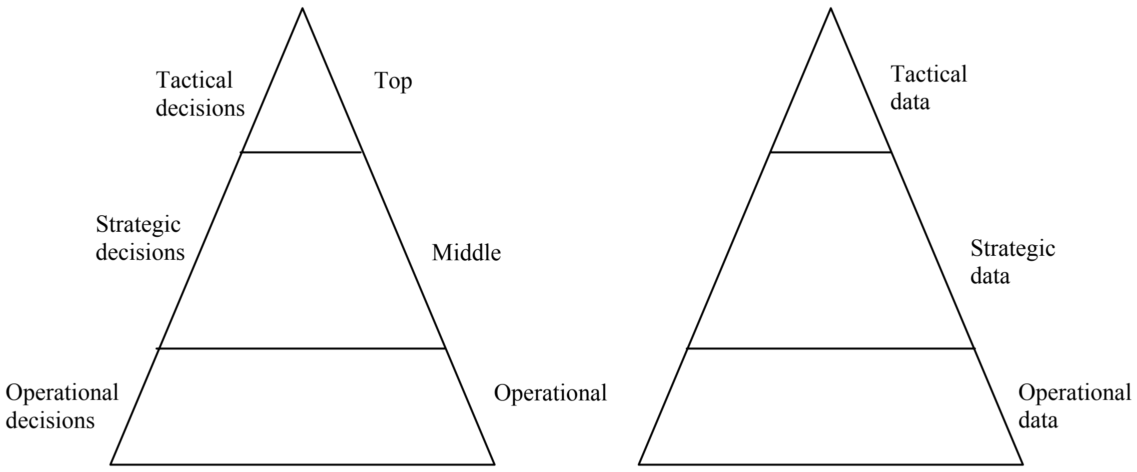

Figure 2.1: Organisational paradigms

2.5 ORGANISATION, INFORMATION AND DECISION

The successful development of information systems calls for a deep understanding of the organisational structure and dynamics of the enterprise. Some organisation are goal oriented and the analyst must be clear as to what information exactly needs to be collected, stored and analysed. Since every information must have a context, only operational information that ultimately has some decision making contribution must be collected. Secondly, the information collected and processed must be consistent with the level of the organisation to which it is to be presented.

One of the classifications indicate that there are basically three levels of management, independent of the size of the enterprise: operational level, middle level and top level management. Operational decisions call for large volumes of internal data (local to the enterprise). The middle management is concerned with medium range (tactical) decisions that call for much less information. The top management is concerned with long-term (strategic) decisions calling for just a few vital pieces of internal information but a lot of external information as well. Any successful information system should take into account, such a pattern of information needs by the management. This is generally pictorially displayed in the form of Management *vs.* Information Pyramid.

The importance of information to management is further emphasised by the fact that much of the management primarily takes decisions. While there are several views of what constitutes management, it is generally accepted that planning, organising, coordinating, directing and control are all concerned with decision making. In this text, we take such a view of management and we perceive management information systems to support such managerial decision making. We also would like to emphasise that information systems should address clearly the situations of programmed decisions and non-programmed decisions by properly structuring the appropriate information. Failure to recognise intrinsic differences may lead to a failure of the information system.

2.6 PROBLEM IDENTIFICATION

We have to reduce the initial, complex problem to a series of simple tasks, each of which can be solved fairly easily. Secondly, by working at several levels, we can move from an overall outline of a solution to various details, without having these details interfere with our overall understanding of the solution. Thirdly, for large problems, we may be able to assign various parts of the overall solution to different people. Then, the overall structure will allow us to combine various pieces into the final solution of the overall problem.

Principles of Coupling, Cohesion, and information Hiding

Often there may be several reasonable ways to divide an overall solution into modules, and it may be difficult to determine which approach will work best. In any decomposition into modules, however, a few general principles should be considered. First, we must try to keep the steps independent so that these steps do not interfere with each other. Each module should do a specific task, but we should try to limit the interaction among various modules. More formally, we can consider the coupling of modules in the problem design. Here, coupling describes the amount that modules depend on each other, and the decomposition of large problems into pieces should minimize coupling. The manner and degree of interdependence between software modules is known as Coupling.

In software engineering, the relation between various parts of a module to a central theme is called cohesion, and we should strive to write concise, cohesive modules for each part of the solution outline. The manner and degree to which the tasks performed by a single software module are related to one another is known as Cohesion.

The third principle related to module decomposition involves the concept of abstraction, where we try to separate the details of a step from the way that step fits in the solution as a whole. Of course, at some point, all details must be specified, but once they have been determined, we do not want to worry about them when that task is performed in the overall solution. When we work with major steps in a solution, we should not be concerned about just how those steps are actually done. Rather, we concentrate on putting those individual tasks into an appropriate framework. This is the concept of information hiding, where the details of a task at one level are hidden from the use of that task at a higher level. From this standpoint, we normally should write modules for different levels of detail in a solution. For example, one main module often controls the overall running of a program, and that module calls on other modules to perform separate major tasks.

Similarly, details about data storage, retrieval, and processing can be separated from the logical interactions involving that data. Information hiding, therefore, can be applied to both the structure of problems and the organisation and manipulation of data.

Clearly, a program is correct and useful only if it solves the problem at hand. The development of specifications, a general design, detailed algorithms, and programs, and each of these steps build on the previous ones. Thus, any errors (such as omissions, inconsistencies) from one step normally will be reflected in subsequent work. More specifically, we should analyse and check each problem solving step for errors or potential problems.

Always, it is advisable to follow a particular standard for software development.

2.7 SOFTWARE CRISIS

The transition of a familiarity with software to the development of useful application is not a straight forward task. This has led to the search for methods and techniques to be able to cope with the ever expanding demands for software.

However, it is still useful and desirable to have some feel for the kinds of problems which the programmer and the user faces, collectively perceived as the software crisis.

Software crisis can be broadly classified into the following categories:

2.7.1 From Programmer's Point of View

The following types of problems may contribute in maximum cases to a software crisis:

- Problem of compatibility.
- Problem of portability.
- Problem in documentation.
- Problem in coordination of work of different people where a team is trying to develop software.
- Problems that arise during actual run time in the organisation. Sometimes, the errors are not detected during sample run.
- Problem of piracy of software.
- Customers normally expand their specifications after program design and implementation has taken place.
- Problem of maintenance.

2.7.2 From User's Point of View

There are many sources of problems that arise out of the user's end. Some of these are as follows:

- How to choose a software?
- How to ensure which software is compatible with the hardware specifications?
- The software packages generally does not meet total requirements
- Problem of virus
- Problem of software bugs, which comes to knowledge of customer after considerable data entry
- Certain software run only on specific operating system environment
- The problem of compatibility for user may be because of different size and density of floppy diskettes
- Problem in learning all the facilities provided by the software because companies give only selective information in manual
- Certain software run and create files which expand their used memory spaces and create problems of disk management
- Software crisis develops when system memory requirement of software is more than the existing requirements and/or availability
- Problem of different versions of software
- Security problem for protected data in software.

2.8 ROLE OF A SYSTEMS ANALYST

A systems analyst is a person who conducts a study, identifies activities and objectives and determines a procedure to achieve the objectives. Designing and implementing systems to suit organisational needs are the functions of the systems analyst. He plays a major role in visualising business benefit from computer technology. The analyst is a person with unique skills. He uses these skills to coordinate the efforts of different types of persons in an organisation to achieve business goals.

What a Systems Analyst does?

A systems analyst carries out the following job:

(a) The first and perhaps the most difficult task of the systems analyst is problem definition. Business problems are quite difficult to define. It is also true that problems cannot be solved until they are precisely and clearly defined.

(b) Initially, a systems analyst does not know how to solve a specific problem. He must consult with managers, users and other data processing professionals in defining problems and developing solutions. He uses various methods for data gathering to get the correct solution of a problem.

(c) Having gathered the data relating to a problem, the systems analyst analyses them and thinks of a plan to solve it. He may not always come up personally with the best way of solving a problem but pulls together other people's ideas and refines them until a workable solution is achieved.

(d) Systems analysts coordinate the process of developing solutions. Since many problems have a number of solutions, the systems analyst must evaluate the merit of such proposed solutions before recommending one to the management.

(e) Systems analysts are often referred to as planners. A key part of the systems analyst's job is to develop a plan to meet the management's objectives.

(f) When the plan has been accepted, the systems analyst is responsible for designing it so that management's goal could be achieved. Systems design is a time consuming, complex and precise task.

(g) Systems must be thoroughly tested. The systems analyst often coordinates the testing procedures and helps in deciding whether or not the new system is meeting standards established in the planning phase.

Attributes of an effective Systems Analyst

A systems analyst must have the following attributes:

(a) **Knowledge of people:** Since a systems analyst works with others so closely, s/he must understand their needs and what motivates them to develop systems properly.

(b) **Knowledge of Business functions:** A systems analyst must know the environment in which s/he works. He must be aware of the peculiarities of management and the users at his installation and realise how they react to systems. An analyst should have working knowledge of accounting and marketing principles is a must since so many systems are built around these two areas. He must be familiar with his company's product and services and management's policies in areas concerning him.

☞ Check Your Progress 2

1) The manner and degree of interdependence between software modules is known as _________.

2) The manner and degree to which the tasks performed by a single software module are related to one another is known as ___________.

2.9 SUMMARY

The manager who forgets that software engineering work is an intensely human endeavour will never have success in project management. A manager who fails to encourage comprehensive customer communication early in the evolution of a project risks building an elegant solution for a different problem. Finally, the manager who pays little attention to the process runs the risk of inserting competent technical methods and tools into a vacuum.

First, we must try to keep the steps independent so that these steps do not interfere with each other. Each module should do a specific task, but we should try to limit the interaction among various modules. More formally, we can consider the coupling of modules in the problem design. Here, coupling describes the amount that modules depend on each other, and the decomposition of large problems into pieces should minimize coupling. The manner and degree of interdependence between software modules is known as Coupling.

In software engineering, the relation between various parts of a module to a central theme is called cohesion, and we should strive to write concise, cohesive modules for each part of the solution outline. The manner and degree to which the tasks performed by a single software module are related to one another is known as Cohesion.

The third principle related to module decomposition involves the concept of abstraction, where we try to separate the details of a step from the way that step fits in the solution as a whole. Of course, at some point, all details must be specified, but once they have been determined, we do not want to worry about them when that task is performed in the overall solution. When we work with major steps in a solution, we should not be concerned about just how those steps are actually done. Rather, we concentrate on putting those individual tasks into an appropriate framework. This is the concept of information hiding, where the details of a task at one level are hidden from the use of that task at a higher level. From this standpoint, we normally should write modules for different levels of detail in a solution. For example, one main module often controls the overall running of a program, and that module calls on other modules to perform separate major tasks.

2.10 SOLUTIONS/ANSWERS

Check Your Progress 1

1) people, problem and process.

2) tasks, milestones and deliverables.

Check Your Progress 2

1) Coupling

2) Cohesion

2.11 FURTHER READINGS

1) *Software Engineering, Sixth Edition,* Ian Sommerville; Pearson Education.

2) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

Reference websites

- **http://www.rspa.com**
- **http://www.ieee.org**
- **http://standards.ieee.org**