

Module 12

Software Project Monitoring and Control

Lesson 30

Organization and Team Structures

Specific Instructional Objectives

At the end of this lesson the student would be able to:

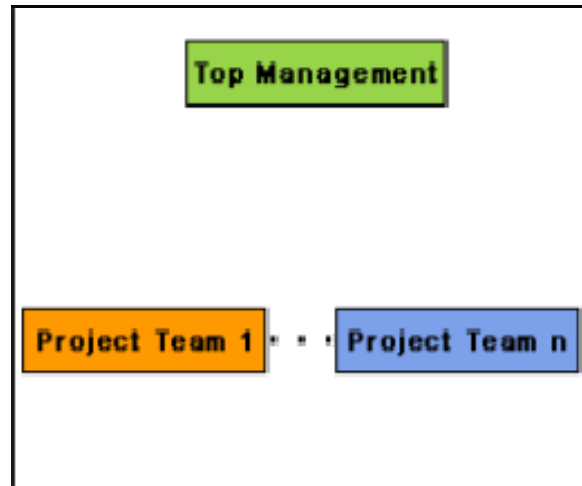
- I Explain the necessity of a suitable organization structure.
- Differentiate between functional format and project format in the context of organization structure.
- Identify the advantages of a functional organization over a project organization.
- Explain why the functional format is not suitable for small organizations handling just one or two projects.
- Identify the important types of team structures of an organization.
- Explain what is meant by egoless programming technique.
- Identify the characteristics of a good software engineer.

Organization structure

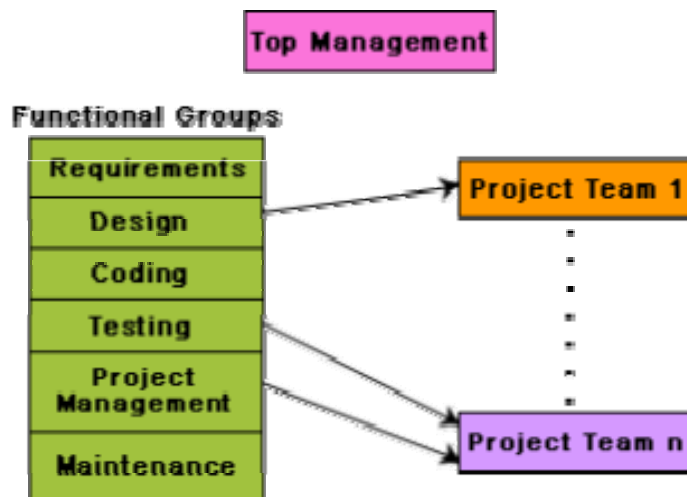
Usually every software development organization handles several projects at any time. Software organizations assign different teams of engineers to handle different software projects. Each type of organization structure has its own advantages and disadvantages so the issue “how is the organization as a whole structured?” must be taken into consideration so that each software project can be finished before its deadline.

Functional format vs. project format

There are essentially two broad ways in which a software development organization can be structured: functional format and project format. In the project format, the project development staff are divided based on the project for which they work (as shown in fig. 12.1). In the functional format, the development staff are divided based on the functional group to which they belong. The different projects borrow engineers from the required functional groups for specific phases to be undertaken in the project and return them to the functional group upon the completion of the phase.



(a) Project Organization



(b) Functional Organization

Fig. 12.1: Schematic representation of the functional and project organization

In the functional format, different teams of programmers perform different phases of a project. For example, one team might do the requirements specification, another do the design, and so on. The partially completed product passes from one team to another as the project evolves. Therefore, the functional format requires considerable communication among the different teams because the work of one team must be clearly understood by the subsequent teams working on the project. This requires good quality documentation to be produced after every activity.

In the project format, a set of engineers is assigned to the project at the start of the project and they remain with the project till the completion of the project. Thus, the same team carries out all the life cycle activities. Obviously, the functional format requires more communication among teams than the project format, because one team must understand the work done by the previous teams.

Advantages of functional organization over project organization

Even though greater communication among the team members may appear as an avoidable overhead, the functional format has many advantages. The main advantages of a functional organization are:

- Ease of staffing
- Production of good quality documents
- Job specialization
- Efficient handling of the problems associated with manpower turnover.

The functional organization allows the engineers to become specialists in particular roles, e.g. requirements analysis, design, coding, testing, maintenance, etc. They perform these roles again and again for different projects and develop deep insights to their work. It also results in more attention being paid to proper documentation at the end of a phase because of the greater need for clear communication as between teams doing different phases. The functional organization also provides an efficient solution to the staffing problem. We have already seen that the staffing pattern should approximately follow the Rayleigh distribution for efficient utilization of the personnel by minimizing their wait times. The project staffing problem is eased significantly because personnel can be brought onto a project as needed, and returned to the functional group when they are no more needed. This possibly is the most important advantage of the functional organization. A project organization structure forces the manager to take in almost a constant number of engineers for the entire duration of his project. This results in engineers idling in the initial phase of the software development and are under tremendous pressure in the later phase of the development. A further advantage of the functional organization is that it is more effective in handling the problem of manpower turnover. This is because engineers can be brought in from the functional pool when needed. Also, this organization mandates production of good quality documents, so new engineers can quickly get used to the work already done.

Unsuitability of functional format in small organizations

In spite of several advantages of the functional organization, it is not very popular in the software industry. The apparent paradox is not difficult to explain. The project format provides job rotation to the team members. That is, each team

member takes on the role of the designer, coder, tester, etc during the course of the project. On the other hand, considering the present skill shortage, it would be very difficult for the functional organizations to fill in slots for some roles such as maintenance, testing, and coding groups. Also, another problem with the functional organization is that if an organization handles projects requiring knowledge of specialized domain areas, then these domain experts cannot be brought in and out of the project for the different phases, unless the company handles a large number of such projects. Also, for obvious reasons the functional format is not suitable for small organizations handling just one or two projects.

Team structures

Team structure addresses the issue of organization of the individual project teams. There are some possible ways in which the individual project teams can be organized. There are mainly three formal team structures: chief programmer, democratic, and the mixed team organizations although several other variations to these structures are possible. Problems of different complexities and sizes often require different team structures for chief solution.

Chief Programmer Team

In this team organization, a senior engineer provides the technical leadership and is designated as the chief programmer. The chief programmer partitions the task into small activities and assigns them to the team members. He also verifies and integrates the products developed by different team members. The structure of the chief programmer team is shown in fig. 12.2. The chief programmer provides an authority, and this structure is arguably more efficient than the democratic team for well-understood problems. However, the chief programmer team leads to lower team morale, since team-members work under the constant supervision of the chief programmer. This also inhibits their original thinking. The chief programmer team is subject to single point failure since too much responsibility and authority is assigned to the chief programmer.

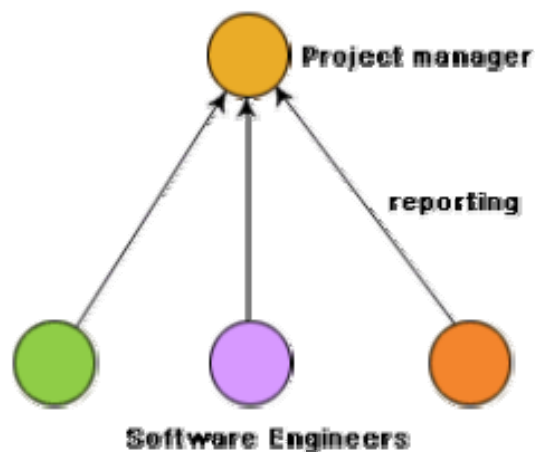


Fig. 12.2: Chief programmer team structure

The chief programmer team is probably the most efficient way of completing simple and small projects since the chief programmer can work out a satisfactory design and ask the programmers to code different modules of his design solution. For example, suppose an organization has successfully completed many simple MIS projects. Then, for a similar MIS project, chief programmer team structure can be adopted. The chief programmer team structure works well when the task is within the intellectual grasp of a single individual. However, even for simple and well-understood problems, an organization must be selective in adopting the chief programmer structure. The chief programmer team structure should not be used unless the importance of early project completion outweighs other factors such as team morale, personal developments, life-cycle cost etc.

Democratic Team

The democratic team structure, as the name implies, does not enforce any formal team hierarchy (as shown in fig. 12.3). Typically, a manager provides the administrative leadership. At different times, different members of the group provide technical leadership.

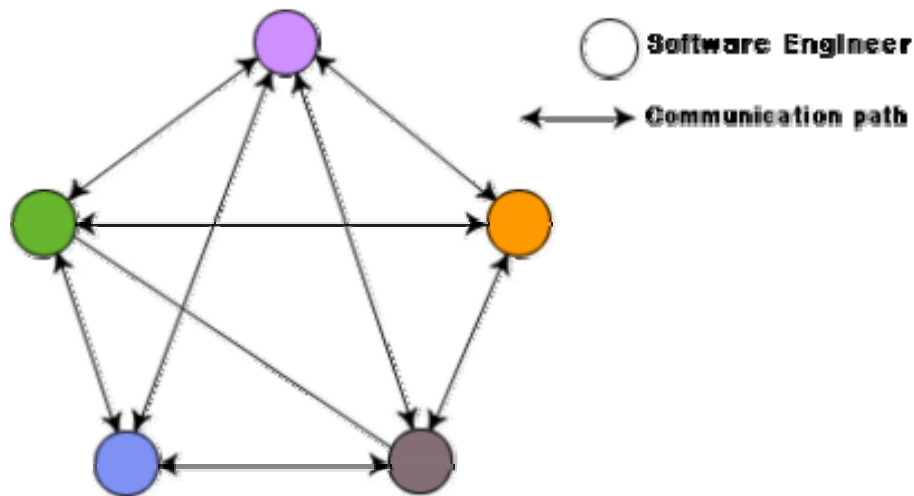


Fig. 12.3: Democratic team structure

The democratic organization leads to higher morale and job satisfaction. Consequently, it suffers from less man-power turnover. Also, democratic team structure is appropriate for less understood problems, since a group of engineers can invent better solutions than a single individual as in a chief programmer team. A democratic team structure is suitable for projects requiring less than five or six engineers and for research-oriented projects. For large sized projects, a pure democratic organization tends to become chaotic. The democratic team organization encourages egoless programming as programmers can share and review one another's work.

Mixed Control Team Organization

The mixed team organization, as the name implies, draws upon the ideas from both the democratic organization and the chief-programmer organization. The mixed control team organization is shown pictorially in fig. 12.4. This team organization incorporates both hierarchical reporting and democratic set up. In fig. 12.4, the democratic connections are shown as dashed lines and the reporting structure is shown using solid arrows. The mixed control team organization is suitable for large team sizes. The democratic arrangement at the senior engineers level is used to decompose the problem into small parts. Each democratic setup at the programmer level attempts solution to a single part. Thus, this team organization is eminently suited to handle large and complex programs. This team structure is extremely popular and is being used in many software development companies.

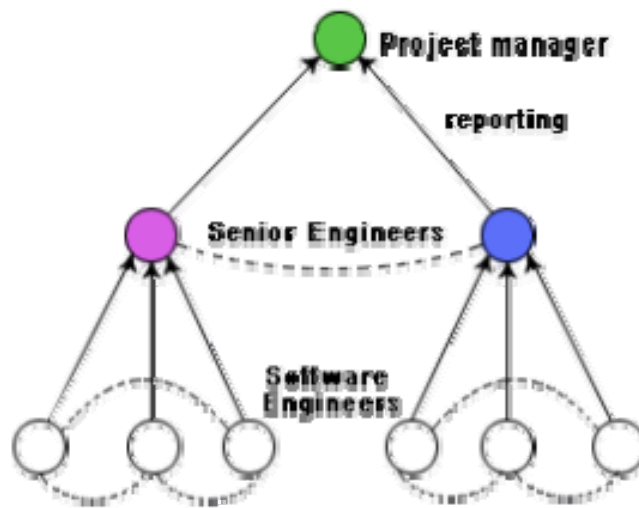


Fig. 12.4: Mixed team structure

Egoless programming technique

Ordinarily, the human psychology makes an individual take pride in everything he creates using original thinking. Software development requires original thinking too, although of a different type. The human psychology makes one emotionally involved with his creation and hinders him from objective examination of his creations. Just like temperamental artists, programmers find it extremely difficult to locate bugs in their own programs or flaws in their own design. Therefore, the best way to find problems in a design or code is to have someone review it. Often, having to explain one's program to someone else gives a person enough objectivity to find out what might have gone wrong. This observation is the basic idea behind code walk throughs. An application of this, is to encourage a

democratic team to think that the design, code, and other deliverables to belong to the entire group. This is called egoless programming technique.

Characteristics of a good software engineer

The attributes that good software engineers should possess are as follows:

- Exposure to systematic techniques, i.e. familiarity with software engineering principles.
- Good technical knowledge of the project areas (Domain knowledge).
- Good programming abilities.
- Good communication skills. These skills comprise of oral, written, and interpersonal skills.
- High motivation.
- Sound knowledge of fundamentals of computer science.
- Intelligence.
- Ability to work in a team.
- Discipline, etc.

Studies show that these attributes vary as much as 1:30 for poor and bright candidates. An experiment conducted by Sackman [1968] shows that the ratio of coding hour for the worst to the best programmers is 25:1, and the ratio of debugging hours is 28:1. Also, the ability of a software engineer to arrive at the design of the software from a problem description varies greatly with respect to the parameters of quality and time.

Technical knowledge in the area of the project (domain knowledge) is an important factor determining the productivity of an individual for a particular project, and the quality of the product that he develops. A programmer having a thorough knowledge of database application (e.g. MIS) may turn out to be a poor data communication engineer. Lack of familiarity with the application areas can result in low productivity and poor quality of the product.

Since software development is a group activity, it is vital for a software engineer to possess three main kinds of communication skills: Oral, Written, and Interpersonal. A software engineer not only needs to effectively communicate with his teammates (e.g. reviews, walk throughs, and other team communications) but may also have to communicate with the customer to gather product requirements. Poor interpersonal skills hamper these vital activities and often show up as poor quality of the product and low productivity. Software engineers are also required at times to make presentations to the managers and to the customers. This requires a different kind of communication skill (oral communication skill). A software engineer is also expected to document his work (design, code, test, etc.) as well as write the users' manual, training manual, installation manual, maintenance manual, etc. This requires good written communication skill.

Motivation level of software engineers is another crucial factor contributing to his work quality and productivity. Even though no systematic studies have been reported in this regard, it is generally agreed that even bright engineers may turn out to be poor performers when they have lack motivation. An average engineer who can work with a single mind track can outperform other engineers, higher incentives and better working conditions have only limited affect on their motivation levels. Motivation is to a great extent determined by personal traits, family and social backgrounds, etc.