5

People Management and Team Organization

LEARNING OBJECTIVES

- To be aware of the importance of people issues in software development
- To know of different ways to organize work
- To know of major types of management styles
- To appreciate different ways to organize a software development team

Finding the right organizational framework and the right mix of skills for a development team is a difficult matter. Little well-founded theory is available for this. Yet, many stories of successful and less successful projects discern some of the intricacies of project team issues. This chapter sketches the major issues involved.

People are the organization's most important asset
(Humphrey, 1997a)

In most organizations that develop software, programmers, analysts and other professionals work together in a team. An adequate team structure depends on many factors, such as the number of people involved, their experience and involvement in the project, the kind of project, individual differences and style. These factors also influence the way projects are to be managed. In this chapter, we discuss various aspects of people management, as well as some of the more common team organizations for software development projects.

The work to be done within the framework of a project, be it a software development project, building a house, or the design of a new car, involves a number of tasks. A critical part of management responsibility is to coordinate the tasks of all participants.

This coordination can be carried out in a number of ways. There are both external and internal influences on the coordination mechanism. Internal influences originate from characteristics of the project. External influences originate from the project's organizational environment. If these influences ask for conflicting coordination mechanisms, conflicts between the project and the environment are lurking around the corner.

Consider as an example a highly innovative software development project, to be carried out within a government agency. The characteristics of the project may ask for a flexible, informal type of coordination mechanism, where the commitment of specialized individuals, rather than a strict adherence to formal procedures, is a critical success factor. On the other hand, the environment may be geared towards a bureaucracy with centralized control, which tries to impose formal procedures onto project management. These two mechanisms do not work harmoniously. As a consequence, management may get crushed between those opposing forces.

Section 5.1 further elaborates the various internal and external factors that affect the way projects are managed, and emphasizes the need to pay ample attention to the human element in project management.

Software development involves teamwork. The members of the team have to coordinate their work, communicate their decisions, etc. For a small project, the team will consist of up to a few individuals. As the size of the project increases, so will the team. Large teams are difficult to manage, though. Coordinating the work of a large team is difficult. Communication between team members tends to increase exponentially with the size of the team (see also chapter 7). Therefore, large teams are usually split into smaller teams in a way that confines most of the coordination and communication within the sub-team.

Section 5.2 discusses several ways to organize a software development team. Of these, the hierarchical and matrix organizations can be found in other types of business too, while the chief programmer, SWAT and agile team are rather specific to software development. Though open source projects have no means to impose team structure, they usually converge to an onion-like organization as discussed in section 5.2.6.

Because of outsourcing, networked companies and globalization, software development has become a distributed activity. Teams in, say, Amsterdam, Boston and Bangalore may have to cooperate on the development of the same system. How should we split up the tasks between these groups? How to ensure that communication between these groups is effective? Cultural differences play a role as well in multi-site development. For instance, people in Asia respect authority. In Northern America, it is more customary that team members argue with their manager. Managers as well as team members should be aware of those differences, and act accordingly. People issues that affect multi-site software development are discussed in chapter ??.

5.1 People Management

A team is made up of individuals, each of whom has personal goals. It is the task of project management to cast a team out of these individuals, whereby the individual goals are reconciled into one goal for the project as a whole.

Though the individual goals of people may differ, it is important to identify project goals at an early stage, and unambiguously communicate these to the project members. Project members ought to know what is expected of them. If there is any uncertainty in this respect, team members will determine their own goals: one programmer may decide that efficiency has highest priority, another may choose efficient use of memory, while yet a third will decide that writing a lot of code is what counts. Such widely diverging goals may lead to severe problems.

Once project goals are established and the project is under way, performance of project members with respect to the project goals is to be monitored and assessed. This can be difficult, since much of what is being done is invisible and progress is hard to measure.

Ideally, we would like to have an indication of the functionality delivered and define productivity as the amount of functionality delivered per unit of time. Productivity is mostly defined as the number of lines of code delivered per man-month. Everyone will agree that this measure is not optimal, but nothing better has been found. One of the big dangers of using this measure is that people tend to produce as much code as possible. This has a very detrimental effect. The most important cost driver in software development projects is the amount of code to be delivered (see also the chapter on cost estimation). Writing less code is cheaper, therefore, and reuse of existing code is one way to save time and money. It should therefore be strongly advocated. Using the amount of code delivered per man-month as a productivity indicator offers no incentive for software reuse.

Another aspect of people assessment occurs in group processes like peer reviews, inspections and walkthroughs. These techniques are used during verification and validation activities, to discover errors or assess the quality of the code or documentation. In order to make these processes effective it is necessary to clearly separate the documents to be assessed from their authors. Weinberg Weinberg (1971) used the term egoless programming in this Context. An assessment of the product of someone's work should not imply an assessment of that person.

One of the major problems in software development is the coordination of activities of team members. As development projects grow bigger and become more complex, coordination problems quickly accumulate. To counteract these problems, management formalizes communication, for example by having formal project meetings, strictly monitored inspections, and an official configuration control board. However, informal and interpersonal communication is known to be a primary way in which information flows into and through a development organization. It is unwise to rule out this type of communication altogether.[1] Informal, interpersonal communication is most easily accomplished if people are physically at close quarters. Even worse, people are inclined to trade the ease with which information can be obtained against its quality. They will easily accept their neighbor's advice, even if they know that much better advice can be found on the next floor. To counteract this tendency, it is wise to bring together diverse stakeholders in controlled ways, for example by having domain experts in the design team, by having users involved in the testing of software, or through participatory design approaches. The collocation of all stakeholders is a main aspect of agile teams.

Successful software development teams exhibit a mix of qualities: technical competence, end-user empathy, and organization awareness. Technical competency of course is required to deliver a high-quality system in the first place. End-user empathy and organizational awareness have to do with recognition of the individuals and the organization that have to cope with the system. A blend of these orientations in a team helps to ensure sufficient attention is given to each of these aspects (Klein et al. (2002)).

Team management entails a great many aspects, not the least important of which concern the care for the human element. Successes among software development projects can often be traced to a strong focus on cultural and sociological concerns, such as efforts to create a blame-free culture, or the solicitation of commitment and partnership. This chapter touches upon only a few aspects thereof. (Brooks, 1995)

[1]One shining example hereof is the following anecdote from (Weinberg, 1971). The manager of a university computing center got complaints about students and programmers chatting and laughing at the department's coffee machine. Being a real manager, and concerned about productivity, he removed the coffee machine to some remote spot. Quickly thereafter, the load on the computing center consultants increased considerably. The crowd near the coffee machine was in fact an effective, informal communication channel, through which the majority of problems were solved.

and (DeMarco and Lister, 1999) give many insightful observations regarding the human element of software project management.

In the remainder of this section we will confine ourselves to two rather general taxonomies for coordination mechanisms and management styles.

5.1.1 Coordination Mechanisms

In his classic text *Structures in Fives: Designing Effective Organizations*, Mintzberg distinguishes between five typical organizational configurations. These configurations reflect typical, ideal environments. Each of these configurations is associated with a specific coordination mechanism: a preferred mechanism for coordinating the tasks to be carried out within that configuration type. Mintzberg's configurations and associated coordination mechanisms are as follows:

- **Simple structure** In a simple structure there may be one or a few managers, and a core of people who do the work. The corresponding coordination mechanism is called *direct supervision*. This configuration is often found in new, relatively small organizations. There is little specialization, training and formalization. Coordination lies with separate people, who are responsible for the work of others.

- **Machine bureaucracy** When the content of the work is completely specified, it becomes possible to execute and assess tasks on the basis of precise instructions. Mass-production and assembly lines are typical examples of this configuration type. There is little training and much specialization and formalization. The coordination is achieved through *standardization of work processes*.

- **Divisionalized form** In this type of configuration, each division (or project) is granted considerable autonomy as to how the stated goals are to be reached. The operating details are left to the division itself. Coordination is achieved through *standardization of work outputs*. Control is executed by regularly measuring the performance of the division. This coordination mechanism is possible only when the end result is specified precisely.

- **Professional bureaucracy** If it is not possible to specify either the end result or the work contents, coordination can be achieved through *standardization of worker skills*. In a professional bureaucracy, skilled professionals are given considerable freedom as to how they carry out their job. Hospitals are typical examples of this type of configuration.

- **Adhocracy** In projects that are big or innovative in nature, work is divided amongst many specialists. We may not be able to tell exactly what each specialist should do, or how they should carry out the tasks allocated to them. The project's success depends on the ability of the group as a whole to reach a non-specified goal in a non-specified way. Coordination is achieved through *mutual adjustment*.

The coordination mechanisms distinguished by Mintzberg correspond to typical organizational configurations, like a hospital, or an assembly line factory. In his view, different organizations call for different coordination mechanisms. Organizations are not all alike. Following this line of thought, factors external to a software development project are likely to exert an influence on the coordination mechanisms for that project.

Note that most real organizations do not fit one single configuration type. Different parts of one organization may well be organized differently. Also, Mintzberg's configurations represent abstract ideals. In reality, organizations may tend towards one of these configurations, but carry aspects of others as well.

5.1.2 Management Styles

The development of a software system, the building of a house, and the planning of and participation in a family holiday are comparable in that each concerns a coordinated effort carried out by a group of people. Though these projects are likely to be dealt with in widely different ways, the basic assumptions that underlie their organizational structures and management styles have a lot in common.

These basic assumptions can be highlighted by distinguishing between two dimensions in managing people:

– **Relation directedness** This concerns attention to an individual and his relationship to other individuals within the organization.

– **Task directedness** This concerns attention to the results to be achieved and the way in which these results must be achieved.

Both relation and task directedness may be high or low. This leads to four basic combinations, as depicted in figure 5.1. Obviously, these combinations correspond to extreme orientations. For each dimension, there is a whole spectrum of possibilities.

		Task directedness	
		low	high
Relation directedness	low	separation style	commitment style
	high	relation style	integration style

Figure 5.1 Four basic management styles, cf (Reddin, 1970)

The style that is most appropriate for a given situation depends on the type of work to be done:

- **Separation style** This management style is usually most effective for routine work. Efficiency is the central theme. Management acts like a bureaucrat and applies rules and procedures. Work is coordinated hierarchically. Decision-making is top-down, formal, and based on authority. A major advantage of this style is that it results in a stable project organization. On the other hand, real innovations are difficult to accomplish. This style closely corresponds to Mintzberg's coordination through standardization of work processes.

- **Relation style** This style is usually most effective in situations where people have to be motivated, coordinated and trained. The tasks to be performed are bound to individuals. The work is not of a routine character, but innovative, complex, and specialized. Decision-making is a group process; it involves negotiation and consensus building. An obvious weak spot of this style is that it may result in endless chaotic meetings. The manager's ability to moderate efficient decision-making is a key success factor. This style best fits Mintzberg's mutual adjustment coordination mechanism.

- **Commitment style** This is most effective if work is done under pressure. For this style to be effective, the manager has to know how to achieve goals without arousing resentment. Decision making is not done in meetings. Rather, decisions are implied by the shared vision of the team as to the goals of the project. A potential weak spot of this style is that, once this vision has been agreed upon, the team is not responsive to changes in its environment, but blindly stumbles on along the road mapped out. This style best fits Mintzberg's professional bureaucracy.

- **Integration style** This fits situations where the result is uncertain. The work is explorative in nature and the various tasks are highly interdependent. It is the manager's task to stimulate and motivate. Decision-making is informal, bottom-up. This style promotes creativity, and individuals are challenged to get the best out of themselves. A possible weak spot of this style is that the goals of individual team members become disconnected to those of the project, and that they start to compete with one another. Again, Mintzberg's coordination through mutual adjustment fits this situation well.

Each of the coordination mechanisms and management styles identified may be used within software development projects. It is only reasonable to expect that projects with widely different characteristics ask for different mechanisms. For an experienced team asked to develop a well-specified application in a familiar domain, coordination may be achieved through standardization of work processes. For a complex and innovative application, this mechanism is not likely to work, though.

In chapter 8, we will identify various types of software development project and indicate which type of coordination mechanism and management style best fits those projects. It should be noted that the coordination mechanisms suggested in chapter 8 stem from internal factors, i.e. characteristics of the project on hand. As noted before, the project's environment will also exert influence on its organization.

Notice that we looked from the manager to the team and its members in the above discussion. Alternatively, we may look at the relation and task *maturity* of individual team members. Relation maturity concerns the attitude of employees towards their job and management. Task maturity is concerned with technical competence. It is important that the manager aligns his dealings with team members with their respective relation and task maturity. For example, a fresh graduate may have high task maturity and low relation maturity, and so his introduction into a skilled team may warrant some careful guidance.

5.2 Team Organization

Within a team, different roles can be distinguished. There are managers, testers, designers, programmers, and so on. Depending on the size of the project, more than one role may be carried out by one person, or different people may play the same role. The responsibilities and tasks of each of these roles have to be precisely defined in the project plan.

People cooperate within a team in order to achieve an optimal result. Yet it is advisable to strictly separate certain roles. It is a good idea to create a test team that is independent of the development team. Similarly, quality assurance should, in principle, be conducted by people not directly involved in the development process.

Large teams are difficult to manage and are therefore often split up into smaller teams. By clearly defining the tasks and responsibilities of the various sub-teams, communication can be largely confined to communication between members of the same sub-team. Quantifying the cost of interpersonal communication yields insights into effects of team size on productivity and helps to structure large development teams effectively. Some simple formulas for doing so are derived in chapter 7.

In the following subsections we discuss several organizational forms for software development teams.

5.2.1 Hierarchical Organization

In an environment which is completely dedicated to the production of software, we often encounter hierarchical team structures. Depending on the size of the organization or project, different levels of management can be distinguished.

Figure 5.2 gives an example of a hierarchical organization. The rectangles denote the various sub-teams in which the actual work is done. Circled nodes denote managers. In this example, two levels of management can be distinguished. At the lower level, different teams are responsible for different parts of the project. The managers at this level have a primary responsibility in coordinating the work within their respective teams. At the higher level, the work of the different teams is coordinated.

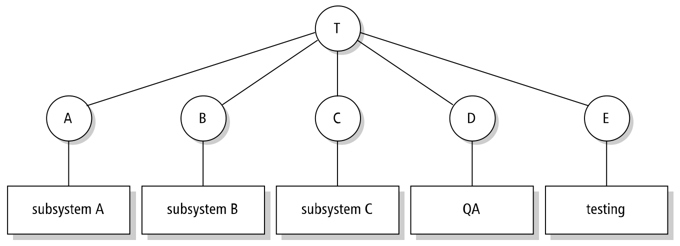

Figure 5.2 A hierarchical team organization

This type of hierarchical organization often reflects the global structure of the system to be developed. If the system has three major subsystems, there may be three teams, one for each subsystem, as indicated in figure 5.2. As indicated in this figure, there may also be functional units associated with specific project-wide responsibilities, such as quality assurance and testing.

It is not possible to associate the hierarchical organization with only one of the coordination mechanisms introduced above. For each unit identified, any one of the coordination mechanisms

mentioned earlier is possible. Also, one need not necessarily apply the same mechanism in each node of the hierarchy. Having different coordination mechanisms within one and the same project is not without problems, though.

Based on an analysis of the characteristics of various subsystems, the respective managers may wish to choose a management style and coordination mechanism that best fits those characteristics. If one or more of the subsystems is highly innovative in nature, their management may opt for a mutual adjustment type of coordination. The higher levels within the hierarchy will usually tend towards a coordination mechanism based on some form of standardization, by imposing rules and procedures as in a machine bureaucracy, or measuring output as in a divisionalized configuration. In such cases, internal and external powers may well clash at one or more of the intermediate levels.

Another critical point in any hierarchical organization is the distance between the top and the bottom of the hierarchical pyramid. The 'real' work is generally done at the lower levels of this pyramid. The people at these lower levels generally possess the real knowledge of the application. The higher one rises in the hierarchy, the less specific the knowledge becomes (this is the main reason why management at these higher levels tends towards coordination through standardization). Yet, most decisions are taken at a fairly high level. In many cases, signals from the lower level somehow get subsumed at one of the intermediate levels.

If information seeps through the various levels in the hierarchy, it tends to become more and more rose-colored. The following scenario is not entirely fictitious:

– bottom: we have severe troubles in implementing module X;

– level 1: there are some problems with module X;

– level 2: progress is steady, I do not foresee any real problems;

– top: everything proceeds according to our plan.

These kinds of distortion are difficult to circumvent altogether. They are, however, reinforced by the fact that the organizational line along which progress is reported is also the line along which the performance of team members is measured and evaluated. Everyone is favored by a positive evaluation and is thus inclined to color the reports accordingly. If data on a project's progress is being collected and processed by people not directly involved in the assessment of team members, you have a much higher chance that the information collected is of sufficient reliability.

An equally problematic aspect of hierarchical organizations lies in the fact that one is judged, both socially and financially, according to the *level* at which one stands within the organization. It is thus natural to aspire to higher and higher levels within the hierarchy. It is, however, not at all clear that this is desirable. The Peter Principle says: in a hierarchical organization each employee in general rises until reaching a level at which he is incompetent. A good programmer need not be a good manager. Good programming requires certain skills. To be a good manager, different skills are needed. In the long run, it seems wiser to maintain people at a level at which they perform well, and reward them accordingly.

5.2.2 Matrix Organization

In an environment where software is a mere byproduct, we often encounter some sort of matrix organization. People from different departments are then allocated to a software development project, possibly part-time. In this type of organization it is sometimes difficult to control progress. An employee has to satisfy several bosses and may have the tendency to play off one boss against another.

We may also use a matrix organization in an environment completely dedicated to software development. The basic unit, then, is a small, specialized group. There may be more than one unit with the same specialization. Possible specializations are, for instance, graphics programming, databases, user interfaces, quality control. The units are organized according to their specialty. Projects, on the other hand, may involve units with different specialties. Individuals are thus organized along two axes, one representing the various specialist groups and one representing the projects to which they are assigned. This type of matrix organization is depicted in figure 5.3.

	real-time programming	graphics	databases	QA	testing
project C	X			X	X
project B	X		X	X	X
project A		X	X	X	X

Figure 5.3 A matrix organization

In such a situation, the project manager is responsible for the successful completion of the project. The manager in charge of one or more units with the same specialty has a longer-term mission, such as maintaining or enlarging the knowledge and expertise of the members of his team. Phrased in terms of the basic management dimensions discussed earlier, the project manager is likely to emphasize task directedness, while the unit manager will emphasize relation directedness. Such an organization can be very effective, provided there is sufficient mutual trust and the willingness to cooperate and pursue the project's goals.

5.2.3 Chief Programmer Team

A team organization known as the chief programmer team was proposed by Harlan Mills around 1970. The kernel of such a team consists of three people. The chief programmer is team leader. He takes care of the design and implements key parts of the system. The chief programmer has an assistant who can stand in for the chief programmer, if needed. Thirdly, a librarian takes care of the administration and documentation. Besides these three people, an additional (small) number of experts may be added to the chief programmer team.

In this type of organization, fairly high demands are made upon the chief programmer. The chief programmer has to be very competent in the technical area, but he also has to have sufficient management capabilities. In other words, are there enough chief programmers? Also, questions of competence may arise. The chief programmer plays a very central role. He takes all the decisions. The other team members may well challenge some of his qualities.

The early notion of a chief programmer team seems somewhat elitist. It resembles a surgeon team in its emphasis on highly specialized tasks and charismatic leadership. The benefits of a team consisting of a small group of peers over huge development teams struggling to produce ever larger software systems may be regained in a modified form of the chief programmer team though.

In this modified form, peer group aspects prevail. The development team then consists of a small group of people collectively responsible for the task at hand. In particular, jobs are not structured around life cycle stages. There are no analysts, designers, or programmers, though the role of tester may be assigned to a specific person. Different levels of expertise may occur within the group. The most experienced persons act as chief programmer and deputy chief programmer, respectively. At the other end of the scale, one or two trainees can be assimilated and get the necessary on-the-job training. A trainee may well act as the team's librarian.

5.2.4 SWAT Team

In projects with an evolutionary or iterative process model such as RAD, a project organization known as the SWAT team is sometimes used. SWAT stands for Skilled With Advanced Tools. We may view the SWAT team as a software development version of a project team in which both task and relation directedness are high.

A SWAT team is relatively small. It typically has four or five members. Preferably, the team occupies one room. Communication channels are kept very short. The team does not have lengthy formal meetings with formal minutes. Rather, it uses workshops and brainstorming sessions of which little more than a snapshot of a white-board drawing is retained.

A SWAT team typically builds incremental versions of a software system. In order to do so effectively, it employs reusable components, very high-level languages, and powerful software generators. The work of team members is supported and coordinated through groupware or workflow management software.

As in the chief programmer team, the leader of a SWAT team is like a foreman in the building industry: he is both a manager and a co-worker. The members of a SWAT team are generalists. They may have certain specialties, but they must also be able to do a variety of tasks, such as participate in a workshop with customers, build a prototype, and test a piece of software.

Team motivation is very important in a SWAT team. A SWAT team often adopts a catchy name, motto or logo. This label then expresses their vision. Individuals derive pride and self-esteem from their membership of a SWAT team.

5.2.5 Agile Team

Agile approaches to software development grew out of, and have a lot in common with, the various iterative development approaches. In the same vein, an agile team has much in common with, e.g., a SWAT team: collocated, short communication channels, a people-oriented attitude rather than a formalistic one. Often, people work in pairs, with a pilot and co-pilot, but without a hierarchy.

Because agile processes have little discipline enforced on them from the outside, they need discipline to come from within the team. Agile teams need self-discipline. If a pair of programmers develops some code and subsequent tests fail, they must take a step back and redo their work. After they have incorporated a piece of work, they must consider the system as a whole and refactor if needed.

For this to succeed, an agile team needs better people than a team that works according to a planning-driven approach. In a planning-driven approach, the plan works like a life-jacket that people can fall back upon. In an agile team, no such life-jacket is available, and people must have swimming skills. In terms of Cockburn's levels of understanding (see Figure 5.4), an agile team requires level 2 or 3 people, and is deemed risky with level 1 people. In planning-driven environments, level 2 or 3 people are only required during the definition stages of development. Thereafter, some level 1 people can be accommodated.

Level	**Description**
Fluent - 3	People at the *fluent* level move flexibly from one approach to another. As software developers, they are able to revise a method to fit an unprecedented new situation
Detaching - 2	People at the *detaching* level are proficient in a single approach, and ready to learn alternatives. They are able to adapt a method to a precedented new situation
Following - 1	People at the *following* level obey a single approach and get confused when confronted with something new. They are able to perform method steps, such as composing a pattern or running tests.

Figure 5.4 Levels of understanding

5.2.6 Open Source Software Development

One of the early books on open source software development is titled *The Cathedral and the Bazaar* (Raymond, 1999). The cathedral refers to traditional, heavyweight, hierarchical software development as is common in closed source software development. Conversely, open source software development is like a bazaar: hordes of anarchist developers casually organized in a virtual networked organization. The bazaar metaphor was chosen to reflect the babbling, chatting, seemingly unorganized form of the middle-Eastern marketplace. Though the bazaar metaphor may fit some open source development groups, many successful open source communities have adopted the more organized onion-like structure depicted in figure 5.5.

In the onion-shaped structure of an open source community, four layers of participation are distinguished:

- The Core Team consists of a small team of experienced developers that also acts as management team. Changes in kernel components of the software can only be made by members of the Core Team.

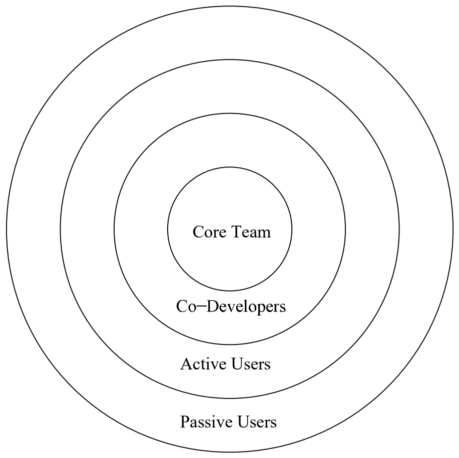

Figure 5.5 Onion shaped structure of an open source community

- The Co-Developers are a larger group of people surrounding the Core Team. Co-Developers review code and do bug fixes.
- The Active Users are users of the most recent release of the system. They submit bug reports and feature requests, but do not themselves program the system.
- Finally, the group of Passive Users is merely using stable releases of the software. They do not interact with the developers.

Usually, outer layers contain more people than inner layers. Often, the Core Team counts no more than 5-15 people. For example, Mockus et al. (2000) reports that the 15-person Core Team of Apache did over 80% of functionality coding.

This type of open source project organization is a meritrocacy; i.e., roles of people are based on talent and proven quality. People in one layer may move up to the next higher layer. Getting to the core is achieved by a process of earning trust, responsibility and status through competition and demonstrated talent. For example, an active user may become co-developer by suggesting quality improvements over a period of time. Likewise, a longstanding record of quality fixes to the code may earn him or her a position in the core team.

The voluntary character of open source development gives rise to some specific challenges:

– Motivation to remain active

– Disagreement between developers

– Communication between developers

An open source community is "a company without walls" (Fang and Neufeld, 2006). People may freely enter and leave the community. Developers participating in open source projects rarely do so selflessly. They expect something in return, such as the ability to learn new things, a higher status within their normal job, attention because they are part of a successful project, and the like. This should come as no surprise. Software professionals have high growth needs (Couger and Zawacki, 1980). Open source projects that challenge developer skills, have a well-modularized code base, and make use of advanced tools have a higher chance of attarcting a sustainable community.

One of the worst things that may happen to an open source project is disagreement between developers. A common obstacle in open source projects is disagreement about development speed Godfrey and Tu (2000). Some developers may want to issue new releases frequently, while others may take a more cautionary stand. Another potential source of disagreement is when users start to feel uncomfortable by the 'undemocratic democracy' of open source projects. Although many people may submit bug fixes or feature requests, the power of what actually happens usually lies with one or a few people in the Core Team. If submissions of a developer get rejected time and again, he may get frustrated and leave the community or decide to create a fork: the developer takes a copy of the source code and starts an independent development track.

Communication between developers is an issue in every distributed team. But in open source projects, the situation is worse because of the floating community membership and the lack of formal documentation. A clear modularization of the code is an important means to reduce the need for extensive communication. Open source communities further tend to use configuration control tools and mailing lists for communication.

5.2.7 General Principles for Organizing a Team

No matter how we try to organize a team, the key point is that it ought to be a *team*. From many tests regarding productivity in software development projects, it turns out again and again that factors concerning team capabilities have a far greater influence than anything else. Factors such as morale, group norms and management style play a more important role than such things as the use of high-level languages, product complexity, and the like (see, for instance, (Lawrence, 1981)).

Some general principles for team organization are given in (Koontz and O'Donnell, 1972). In particular, these general principles also apply to the organization of software development projects:

- **Use fewer, and better, people** Highest productivity is achieved by a relatively small group of people. This holds for novelists, soccer players and bricklayers. There is no reason to believe that it does not equally apply to programmers and other people working in the software field. Also, large groups require more communication, which has a negative effect on productivity and leads to more errors.

- **Try to fit tasks to the capabilities and motivation of the people available** In other words: take care that the Peter Principle does not apply in your situation. In many organizations, excellent programmers can be promoted only into managerial positions. It is far better to also offer career possibilities in the more technical areas of software development and maintenance.

- **In the long run, an organization is better off if it helps people to get the most out of themselves** So you should not pursue either of the following:

  - The reverse Peter Principle: people rise within an organization to a level at which they become indispensable. For instance, a programmer may become the only expert in a certain system. If he does not get a chance to work on anything else, it is not unlikely that this person, for want of a more interesting and challenging task, will leave your organization. At that point, you are in real trouble.
  - The Paul Principle: people rise in an organization to a level at which their expertise becomes obsolete within five years. Given the speed with which new developments enter the market place in software engineering, and computer science in general, it is very important that people get the opportunity to grow and stay abreast of new developments.

- **It is wise to select people such that a well-balanced and harmonious team results** In general, this means that it is not sufficient only to have a few top experts. A soccer team needs regular players as well as stars. Selecting the proper mix of people is a complicated task. There are various good texts available that specifically address this question (for example, (Weinberg, 1971), (Metzger, 1987)).

- **Someone who does not fit the team should be removed** If it turns out that a team does not function as a coherent unit, we are often inclined to wait a little while, see how things develop, and hope for better times to come. In the long run, this is detrimental.

5.3 Summary

Software is written by humans. Their productivity is partly determined by such factors as the programming language used, machine speed, and available tools. The organizational environment in which one is operating is equally important, though. Good team management distinguishes itself from bad team management above all by the degree to which attention is paid to these human factors. The human element in software project management was discussed in section 5.1, together with well-known taxonomies of coordination mechanisms and management styles.

There are different ways to organize software developers in a team. These organizational forms and some of their caveats were discussed in section 5.2. Hierarchical and matrix organizations are not specific to software development, while the chief programmer, SWAT and agile teams originated in the software field. Each of the latter somehow tries to reconcile the two types of management typically required in software development projects: an individualistic, personal approach where one tries to get the best out of team members, and a hierarchical, top-down management style to get things done in time and within budget. Successful open source projects usually have an onion-shaped organization.

5.4 Further Reading

A still very relevant source of information on psychological factors related to software development is (Weinberg, 1971). (Brooks, 1995) and (DeMarco and Lister, 1999) also contain a number of valuable observations. Coordination problems in software development are discussed in (Kraut and Streeter, 1995). (Software, 1996a) and (CACM, 1993b) are special journal issues on managing software projects.

(Mintzberg, 1983) is the classic text on the organization of management. The basic management styles discussed in section 5.1.2 are based on (Reddin, 1970) and (Constantine, 1993).

The chief programmer team is described in (Baker, 1972). Its modified form is described in (Macro and Buxton, 1987). SWAT is discussed in (Martin, 1991). Agile teams are described in, amongst others, (Highsmith, 2004). The three levels of understanding are discussed in (Cockburn, 2002). (Software, 2005) contains a number of articles on how to adopt agile methods.

(SPIP, 2006) contains a collection of articles on open source development processes. Crowston and Howison (2006) discuss the health of open source communities. Aberdour (2007) discusses ways to achieve quality in open source software development.

Exercises

1. Explain Mintzberg's classification of organizational configurations and their associated coordination mechanisms.

2. Discuss Reddin's basic management styles.

3. What are the critical issues in a hierarchical team organization?

4. Highlight the differences between a chief programmer team, a SWAT team and an agile team.

5. Which of Reddin's management styles fits in best with an agile team?

6. What is the Peter Principle? Where does it crop up in software development?

7. Why would an agile team need better people than a team following a planning-based approach?

8. ♡ Consider a software development project you have been involved in. Which style of coordination mechanism or management style best fits this project? Do you consider the management to have been adequate, or does the discussion in section 5.1 point to possible improvements?

9. ♡ From a management point of view, discuss possible pros and cons of having a technical wizard on your development team.

10. ♠ Write an essay on the role of people issues in software development. To do so, you may consult some of the books that focus on people issues in software development, such as (Brooks, 1995), (Weinberg, 1971) or (DeMarco and Lister, 1999).

11. ♠ Discuss the pros and cons of an organization in which the primary departmentalization is vertical (i.e. by specialty, such as databases, human-computer interfaces, or graphics programming) as opposed to one in which the primary departmentalization is horizontal (for example, design, implementation, and testing).

12. ♡ Write an essay on how open source software development projects are managed.

13. ♡ Discuss the pros and cons of letting people rotate between projects from different application domains as opposed to letting them become true experts in one particular application domain.