3

Project Organization and Communication

Two electrical boxes for a rocket, manufactured by different contractors, were connected by a pair of wires. Thanks to a thorough preflight check, it was discovered that the wires had been reversed. After the rocket crashed, the inquiry board revealed that the contractors had indeed corrected the reversed wires as instructed.
In fact, both of them had.

Software engineering is a collaborative activity. The development of software brings together participants from different backgrounds, such as domain experts, analysts, designers, programmers, managers, technical writers, graphic designers, and users. No single participant can understand or control all aspects of the system under development, and thus, all participants depend on others to accomplish their work. Moreover, any change in the system or the application domain requires participants to update their understanding of the system. These dependencies make it critical to share information in an accurate and timely manner.

Communication can take many forms depending on the type of activity it is supporting. Participants communicate their status during regular meetings and record it into meeting minutes. Participants communicate project status to the client during client reviews. The communication of requirements and design alternatives is supported by models and their corresponding documents. Crises and misunderstandings are handled through spontaneous information exchanges such as telephone calls, messages, hallway conversations, and ad hoc meetings. As software engineering projects become large, the time each participant must spend in communication increases, thus decreasing the time spent on technical activities. To address these issues, the organization of projects into teams and the sharing of information through formal and informal channels is essential.

We first describe the basic concepts associated with project organization, such as task, work product, and deliverable. We then describe the communication mechanisms available to participants. Finally, we describe the activities associated with project organization and communication. This chapter is written from the perspective of a project participant (e.g., a developer) who needs to *understand* the project organization and communication infrastructure. The *creation* of the project organization and communication infrastructure is the task of the project manager and is the topic of Chapter 14, *Project Management*.

3.1 Introduction: A Rocket Example

When realizing a system, developers focus on constructing a system that behaves according to specifications. When interacting with other project participants, developers focus on communicating information accurately and efficiently. Even if communication may not appear to be a creative or challenging activity, it contributes as much to the success of the project as a good design or efficient implementation, as illustrated by the following example [Lions, 1996].

Ariane 501

June 4, 1996, 30 seconds into lift-off, Ariane 501, the first prototype of the Ariane 5 series, exploded. The main navigational computer experienced an arithmetic overflow, shut down, and handed control over to its twin backup, as it was designed to do. The backup computer, having experienced the same exception a few hundredths of a second earlier, had already shut down. The rocket, without a navigation system, took a fatal sharp turn to correct a deviation that had not occurred.

An independent board of inquiry took less than 2 months to document how a software error resulted in the massive failure. The navigational system of the Ariane 5 design was one of the few components of Ariane 4 that was reused. It had been flight tested and had not failed for Ariane 4.

The navigation system is responsible for calculating course corrections from a specified trajectory based on input from the inertial reference system. An inertial reference system allows a moving vehicle (e.g., a rocket) to compute its position solely based on sensor data from accelerometers and gyroscopes, that is, without reference to the outside world. The inertial system must first be initialized with the starting coordinates and align its axis with the initial orientation of the rocket. The alignment calculations are done by the navigation system before launch and need to be continuously updated to take into account the rotation of the Earth. Alignment calculations are complex and take approximately 45 minutes to complete. Once the rocket is launched, the alignment data are transferred to the flight navigational system. By design, the alignment calculations continue for another 50 seconds after the transfer of data to the navigation system. The decision enables the countdown to be stopped after the transfer of alignment data takes place but before the engines are ignited without having to restart the alignment calculations (that is, without having to restart a 45-minute calculation cycle). In the event the launch succeeds, the alignment module just generates unused data for another 40 seconds after lift-off.

The computer system of Ariane 5 differed from Ariane 4. The electronics were doubled: two inertial reference systems to compute the position of the rocket, two computers to compare the planned trajectory with the actual trajectory, and two sets of control electronics to steer the rocket. If any component would fail, the backup system would take over.

The alignment system, designed for onground calculations only, used 16-bit words to store horizontal velocity (more than enough for displacements due to the wind and to the rotation of the earth). Thirty seconds into flight, the horizontal velocity of Ariane 5 caused an overflow, raised an exception that was handled by shutting down the onboard computer and handing control to the backup system.

Discussion. The alignment software had not been adequately tested. Although it had been subjected to thousands of tests, none included an actual trajectory. The navigation system was tested individually. Tests were specified by the system team and executed by the builders of the navigation system. The system team did not realize that the alignment module could cause the main processor to shut down, especially not in flight. The component team and the system team had failed to communicate.

In this chapter, we discuss organizational and communication issues within a software project. This topic is not specific to software engineering. Communication is, however, pervasive throughout a software development project. Communication failure is costly and can have a high, and sometimes fatal, impact on the project and the quality of the delivered system.

3.2 An Overview of Projects

The techniques and notations we presented in Chapter 2, *Modeling with UML*, enable project participants to build models of the system and communicate about them. However, system models are not the only information needed when communicating in a project. For example, developers need to know

- Who is responsible for which part of the system?
- Which part of the system is due by when?
- Who should be contacted when a problem with a specific version of a component is discovered?
- How should a problem be documented?
- What are the quality criteria for evaluating the system?
- In which form should new requirements be communicated to developers?
- Who should be informed of new requirements?
- Who is responsible for talking to the client?

Although these questions can be relatively easy answered when all participants share a coffee break in the afternoon, the development of large software systems usually does not succeed with such an ad hoc approach. From a developer's perspective, a project consists of four components (Figure 3-1):

- **Work product.** This is any item produced by the project, such as a piece of code, a model, or a document. Work products produced for the client are called **deliverables**.
- **Schedule.** This specifies when work on the project should be accomplished.
- **Participant.** This is any person participating in a project. Sometimes we also call the participant **project member**.

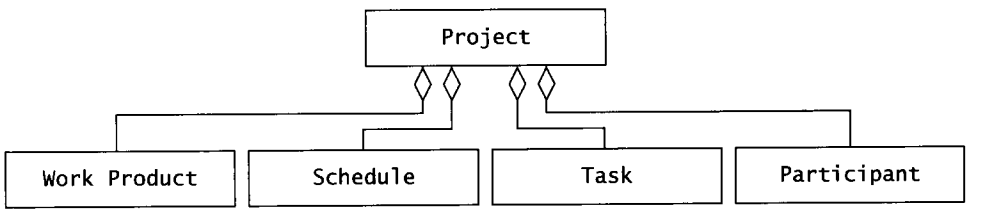

Figure 3-1 Model of a project (UML class diagram).

- **Task.** This is the work to be performed by a project participant to create a work product.

Projects can be defined formally or informally. A signed contract between you and a client requiring the delivery of a software system in three months for one million dollars defines a project; an informal promise you make to your friend to install a new software release on her computer by next week defines a project as well.

Projects come in different types and sizes. Sometimes the characterization of the project type is by the nature of the deliverable. If the outcome is a software system, the project is usually called a software project; building a space shuttle system is called a system project. Projects also come in quite different sizes. Installing a new a space shuttle system, with costs of more than $10 billion and a duration of 10 to 15 years, is a large project, where as changing the furniture of your room is a small project.

From a dynamic point of view, a project can be in any of several phases shown in Figure 3-2. During the **project definition phase**, the project manager, a possible client, and a key project member, the software architect, are involved. The two areas of focus during this phase are an initial understanding of the software architecture, in particular the subsystem decomposition, and the project, in particular the schedule, the work to be performed, and the resources required to do it. This is documented in three documents: the problem statement, the initial software architecture document, and the initial software project management plan. During the **project start phase**, the project manager sets up the project infrastructure, hires participants, organizes them in teams, defines major milestones, and kicks off the project.

During the project definition and project start phases, most decisions are made by the project manager. During the **project steady state phase**, the participants develop the system. They report to their team leader, who is responsible for tracking the status of the developers and identifying problems. The team leaders report the status of their team to the project manager, who then evaluates the status of the complete project. Team leaders respond to deviations from

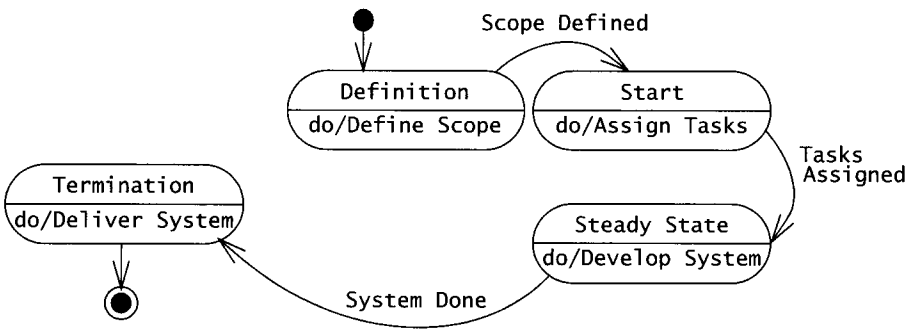

Figure 3-2 States in a software project (UML state machine diagram).

the plan by reallocating tasks to developers or obtaining additional resources from the project manager. The project manager is responsible for the interaction with the client, obtaining formal agreement and renegotiating resources and deadlines.

During the **project termination phase**, the project outcome is delivered to the client and the project history is collected. Most of the developers' involvement with the project ends before this phase. A handful of key developers, the technical writers, and the team leaders are involved with wrapping up the system for installation and acceptance and collecting the project history for future use.

Communication within a project occurs through planned and unplanned events. Planned communication includes:

- **problem inspection**, during which developers gather information from the problem statement, the client, and the user about their needs and the application domain
- **status meetings**, during which teams review their progress
- **peer reviews**, during which team members identify defects and find solutions in preliminary work products
- **client and project reviews**, during which the client or project members review the quality of a work product, in particular deliverables
- **releases**, during which project participants make available to the client and end users versions of the system and its documentation.

Unplanned communication includes:

- **requests for clarification**, during which participants request specific information from others about the system, the application domain, or the project
- **requests for change**, during which participants describe problems encountered in the system or new features that the system should support
- **issue resolution**, during which a conflict between different stakeholders is identified, solutions explored and negotiated, and a resolution agreed upon.

Planned communication helps disseminate information that targeted participants are expected to use. Unplanned communication helps deal with crises and with unexpected information needs. All three communication needs must be addressed for project participants to communicate accurately and efficiently.

When a developer joins a project during the start phase, a problem statement already exists; project management has already written an initial plan to attack the problem, set up a project organization, defined planned communication events, and provided an infrastructure for planned and unplanned communication. Most of the developer's effort when joining a project is to understand these documents and join the existing organizational and communication structures. This is addressed by the following activities:

- *Attend the kick-off meeting.* During this activity, the project participants hear from the client about the problem to be solved and the scope of the system to be developed. This helps them to get a high-level understanding of the problem, which serves as a basis for all other activities.
- *Join a team.* The project manager has decomposed the project into work for individual teams. Participants are assigned to a team based on their skills and interests.
- *Attend training sessions.* Participants who do not have skills for required tasks receive additional training.
- *Join communication infrastructure.* Participants join the project communication infrastructure that supports both planned and unplanned communication events. The infrastructure includes a collection of mechanisms such as groupware, address books, phone books, E-mail services, and video conferencing equipment.
- *Extend communication infrastructure.* Additional bulletin boards and team portals are established specifically for the project.
- *Attend first team status meeting.* During this activity, project participants are taught to conduct status meetings, record status information, and disseminate it to other members of the project.
- *Understand the review schedule.* The review schedule contains a set of high-level milestones to communicate project results in the form of reviews to the project manager and to the client. The objective of project reviews is to inform the project participants of the other teams' status and to identify open issues. The objective of client reviews is to inform the client about the status of the project and to obtain feedback.

In the following sections, we examine these concepts and activities in detail. In Section 3.3, we describe a team-based project organization. In Section 3.4, we discuss the concepts related to project communication. In Section 3.5, we detail the project start activities of a typical team member. In Section 3.6, we provide references to further reading on this topic.

In this chapter, we focus on the perspective of a developer joining a software project, so we do not describe the activities needed to create and manage a project organization and communication infrastructure. We cover these topics in later chapters. Chapter 12, *Rationale Management*, discusses topics related to identifying, negotiating, resolving, and recording issues. Chapter 13, *Configuration Management*, discusses topics related to managing versions, configurations, and releases of documents and system components. In Chapter 14, *Project Management*, we revisit project organization and communication issues from the perspective of the project manager.

3.3 Project Organization Concepts

In this section, we define the following concepts:

- Project Organizations (Section 3.3.1)
- Roles (Section 3.3.2)
- Tasks and Work Products (Section 3.3.3)
- Schedule (Section 3.3.4).

3.3.1 Project Organizations

An important part of any project organization is to define the relationships among participants and between them and tasks, schedule, and work products. In a **team-based organization** (Figure 3-3), the participants are grouped into teams, where a **team** is a small set of participants working on the same activity or task. We distinguish teams from other sets of people, in particular groups and committees. A *group*, for example, is a set of people who are assigned a common task, but they work individually without any need for communication to accomplish their part of the task. A *committee* is comprised of people who come together to review and critique issues and propose actions.

Figure 3-4 shows an instance diagram of an organization for a simple software project consisting of a management team and three developer teams.

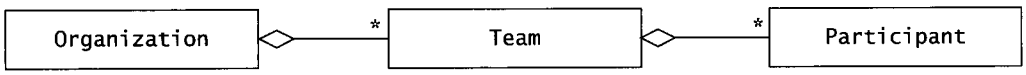

Figure 3-3 A team-based organization consists of organizational units called teams, which consist of participants or other teams (UML class diagram).

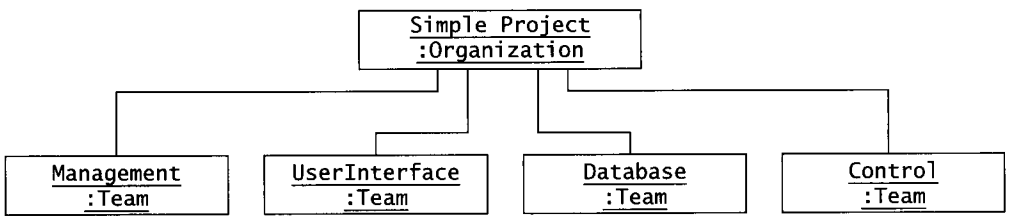

Figure 3-4 Example of a simple project organization (UML instance diagram). Reporting, deciding, and communicating are all made via the aggregation association of the organization.

Project participants interact with each other. The three major types of interaction in a project are:

- **Reporting**. This type of interaction is used for reporting status information. For example, a developer reports to another developer that an API (Application Programmer Interface) is ready, or a team leader reports to a project manager that an assigned task has not yet been completed.
- **Decision**. This type of interaction is used for propagating decisions. For example, a team leader decides that a developer has to publish an API, a project manager decides that a planned delivery must be moved up in time. Another type of decision is the resolution of an issue.
- **Communication**. This type of interaction is used for exchanging all the other types of information needed for decision or status. Communication can take many flavors. Examples are the exchange of requirements or design models or the creation of an argument to support a proposal. An invitation to eat lunch is also a communication.

We call the organization **hierarchical** if both status and decision information are unidirectional; that is, decisions are always made at the root of the organization and passed via the interaction association to the leaves of the organization. Status in hierarchical organizations is generated at the leaves of the organization and reported to the root via the interaction association. The structure of the status and decision information flow is often called the **reporting structure** of the organization. Figure 3-5 illustrates the reporting structure in a hierarchical team-based organization.

In hierarchical organizations, such as a military, the reporting structure also accomplishes the exchange of communication needs. In complex software projects, however, using the existing reporting structure for communication causes many problems. For example, many technical decisions need to be made locally by the developers, but depend on information from developers in other teams. If this information is exchanged via the established reporting

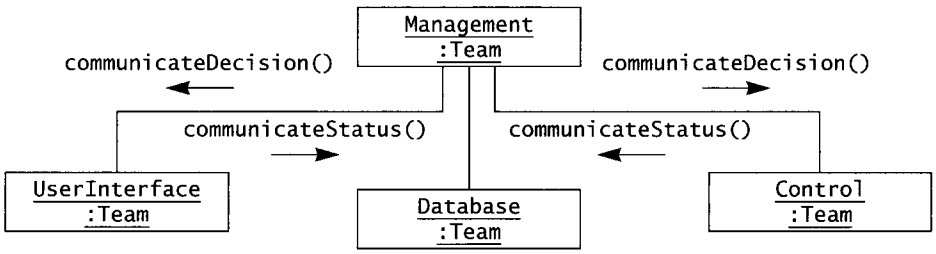

Figure 3-5 Example of reporting structure in a hierarchical organization (UML communication diagram). Status information is reported to the project manager, and corrective decisions are communicated back to the teams by the team leaders. The team leaders and the project manager are called the management team.

structure, the decision-making process can be slowed significantly. Even worse, it often leads to garbling of the information, given its complexity and volume.

The solution to this problem is to exchange information via an additional communication structure that allows participants to communicate directly with each other and in ways different from the reporting structure. Often the communication is delegated to a developer, called a **liaison**, who is responsible for shuttling information back and forth.

Figure 3-6 depicts an example of an organization with liaisons and additional communication lines that deviate from the reporting structure. The documentation team, for example, has a liaison to the user interface team to facilitate information about recent changes made to the appearance of the system. Teams that do not work directly on a subsystem, but rather work on a task that crosses the subsystem team organization, are called **cross-functional teams**. Examples of cross-functional teams include the documentation team, the architecture team, and the testing team.

We call this communication structure, and often also the organization itself, **liaison based**. Liaisons use non-hierarchical communication lines to talk with the liaisons in cross-functional teams. In liaison-based communication structures, the responsibility of team leaders is extended by a new task: not only do they have to make sure that the project manager is aware of the status of the team, but also that team members have all the information they need from other teams. This requires the selection of effective communicators as liaisons to ensure that necessary

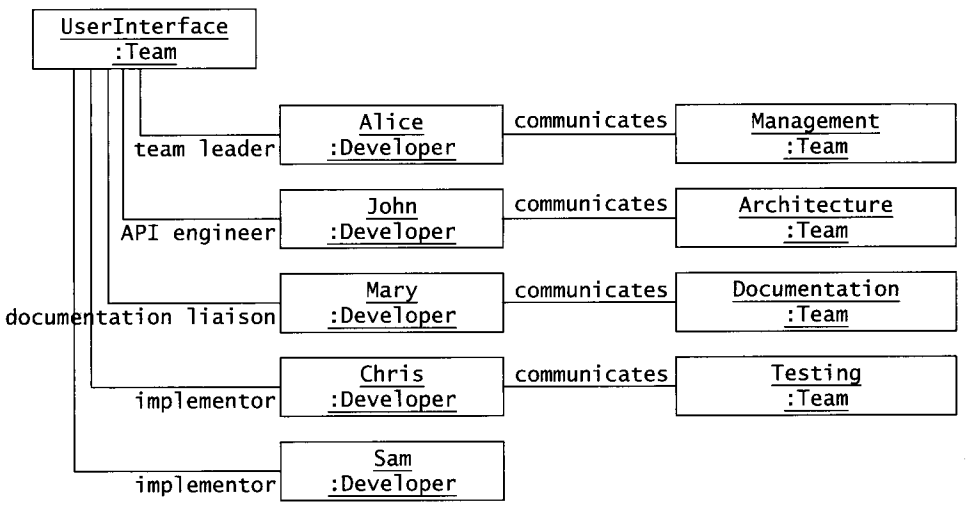

Figure 3-6 Examples of a liaison-based communication structure (UML object diagram). The team is composed of five developers. Alice is the team leader, also called the liaison to the management team. John is the API engineer, also called the liaison to the architecture team. Mary is the liaison to the documentation team. Chris and Sam are implementors and interact with other teams only informally.

communication paths exist. If we allow developers to communicate directly with each other as well, we call the communication structure (and the organization) **peer based**.

3.3.2 Roles

A **role** defines the set of technical and managerial tasks that are expected from a participant or team. In a team-based organization, we assign tasks to a person or a team via a role. For example, the role of tester of a subsystem team consists of the tasks to define the test suites for the subsystem under development, for executing these tests, and for reporting discovered defects back to the developers.

In a software project we distinguish between the following four types of roles: management roles, development roles, cross-functional roles, and consultant roles (Figure 3-7).

Management roles (e.g., project manager, team leader) are concerned with the organization and execution of the project within constraints. We describe this type of role in more detail in Chapter 14, *Project Management*.

Development roles are concerned with specifying, designing, and constructing subsystems. These roles include the analyst, the system architect, the object designer, the implementor, and the tester. Table 3-1 describes examples of development roles in a subsystem team. We describe development roles in more detail in Chapters 5–11.

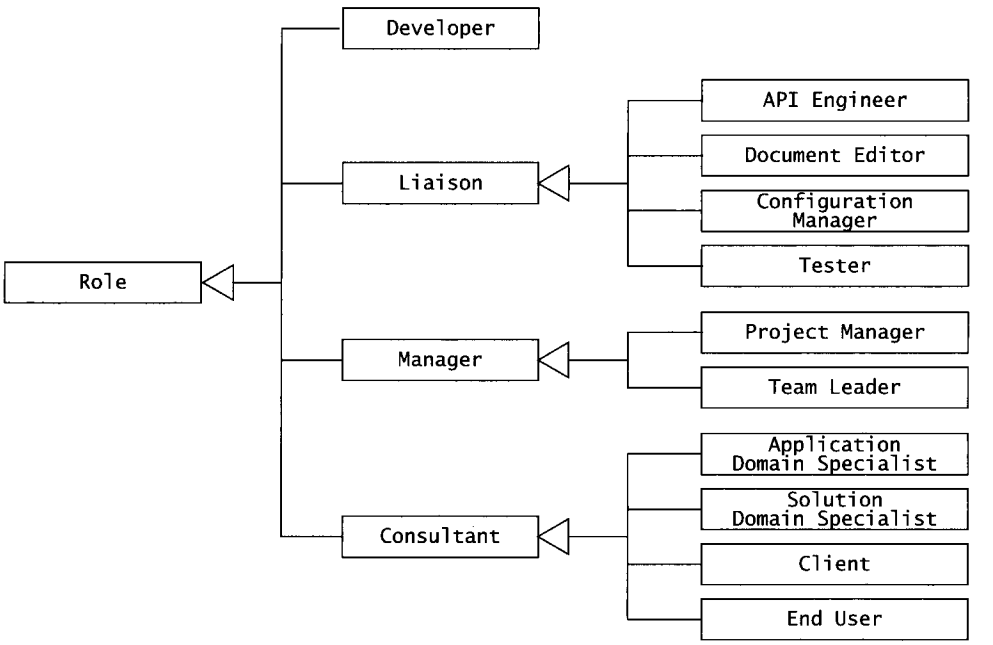

Figure 3-7 Types of roles found in a software engineering project (UML class diagram).