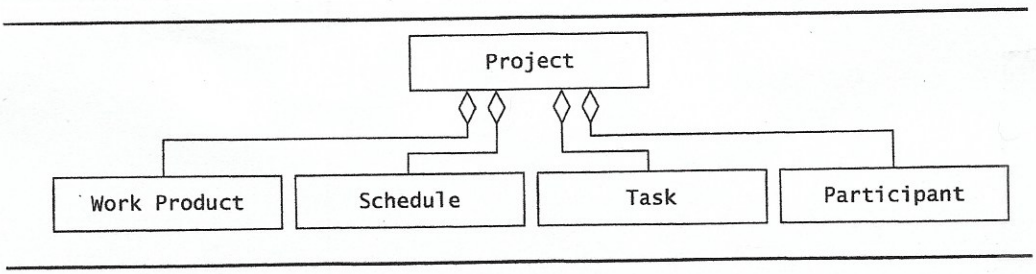

Figure 3-1 Model of a project (UML class diagram).

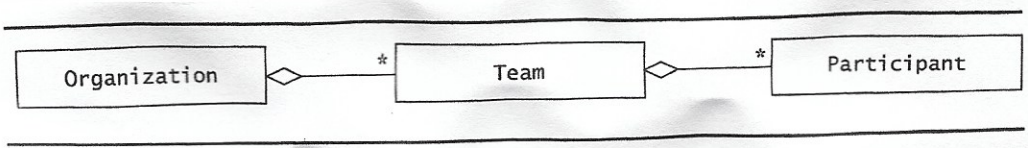

Figure 3-3 A team-based organization consists of organizational units called teams, which consist of participants or other teams (UML class diagram).

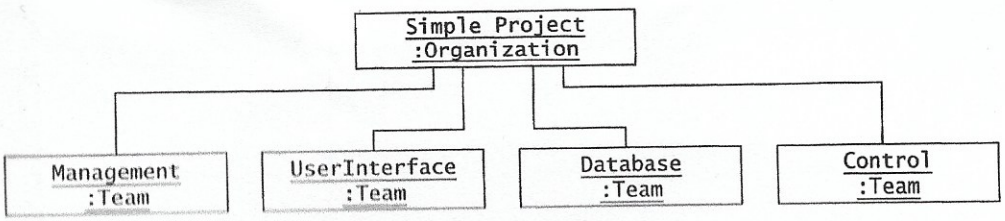

Figure 3-4 Example of a simple project organization (UML instance diagram). Reporting, deciding, and communicating are all made via the aggregation association of the organization.

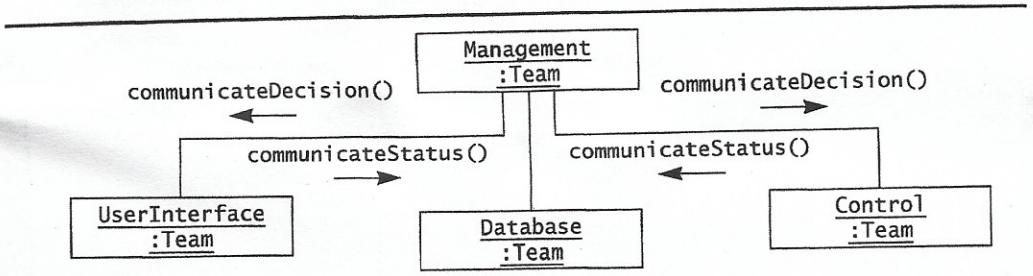

Figure 3-5 Example of reporting structure in a hierarchical organization (UML communication diagram). Status information is reported to the project manager, and corrective decisions are communicated back to the teams by the team leaders. The team leaders and the project manager are called the management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

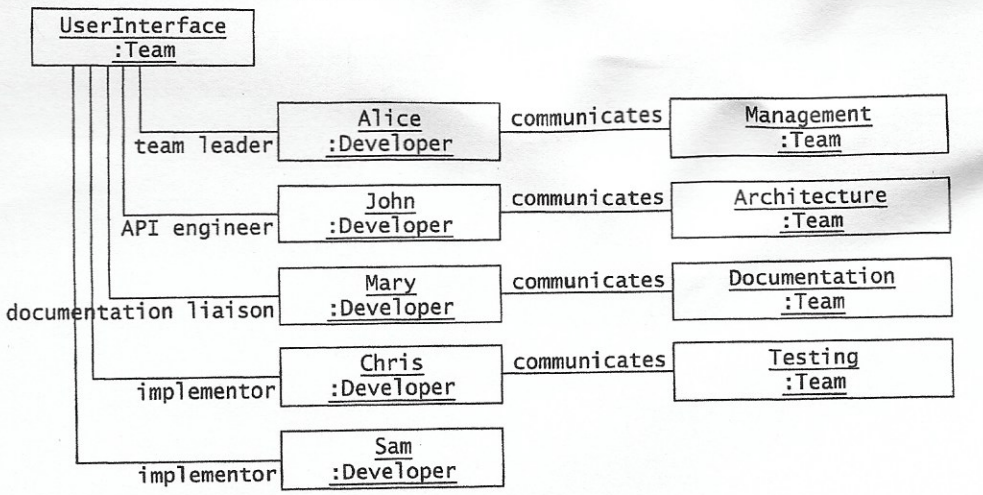

Figure 3-6 Examples of a liaison-based communication structure (UML object diagram). The team is composed of five developers. Alice is the team leader, also called the liaison to the management team. John is the API engineer, also called the liaison to the architecture team. Mary is the liaison to the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

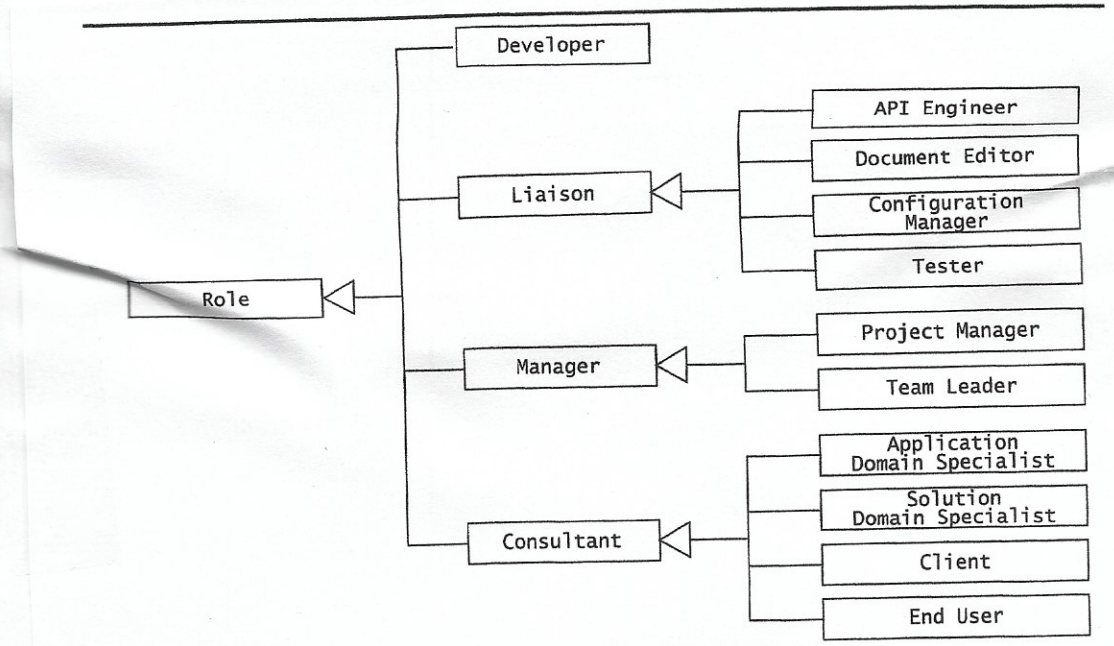

Figure 3-7 Types of roles found in a software engineering project (UML class diagram).

Table 3-1 Examples of roles.

Role	Responsibilities
System architect	The system architect ensures consistency in design decisions and interface styles. The system architect ensures the consistency of the design in the configuration management and testing teams, in particular in the formulation of the configuration management policy as well as the system integration strategy. This is mainly an integration role consuming information from each subsystem team.
Object designer	The object designer is responsible for the interface definition of the assigned subsystem. The interface has to reflect the functionality already assigned to the subsystem and to accommodate the needs of the dependent subsystems. When functionality is traded off with other subsystems, resulting in subsystem changes, the object designer is responsible for propagating changes back to the subsystem team.
Implementor	The implementor is responsible for the coding of a class or a number of classes associated with the subsystem.
Tester	A tester is responsible for evaluating that each subsystem works as specified by the object designer. Often, development projects have a separate team responsible only for testing. Separating the roles of implementor and tester leads to more effective testing.

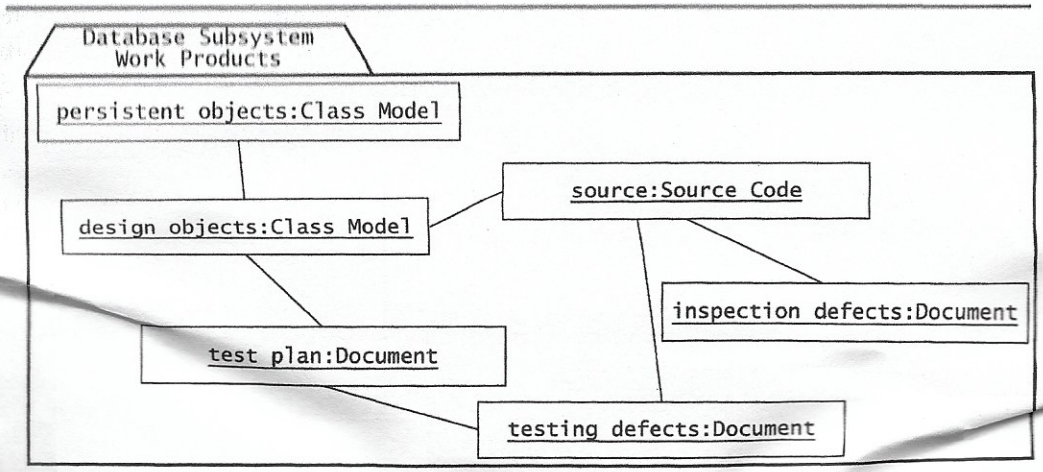

Figure 3-8 Work products for the database subsystem team (UML object diagram). Associations represent dependencies among work products.

Table 3-2 Description of the internal work products depicted in Figure 3-8.

Work product	Type	Description
Persistent Objects	**Class model**	This class model describes completely the objects that are stored by the storage subsystem. For each class, this includes all the attributes, associations, roles, and multiplicities.
Design objects	**Class model**	This class model describes all the objects needed by the storage subsystem that are not described in the persistent object class model.
Subsystem	**Source code**	This is the source code delivered to the testing team.
Test plan	**Document**	This document outlines the test strategy, test criteria, and test cases that are used to find defects in the storage subsystem.
Testing defects	**Document**	This document lists all the defects that have already been found in the storage subsystem through testing.
Inspection defects	**Document**	This document lists all the defects that have already been found in the storage subsystem through peer review, as well as their planned repairs.

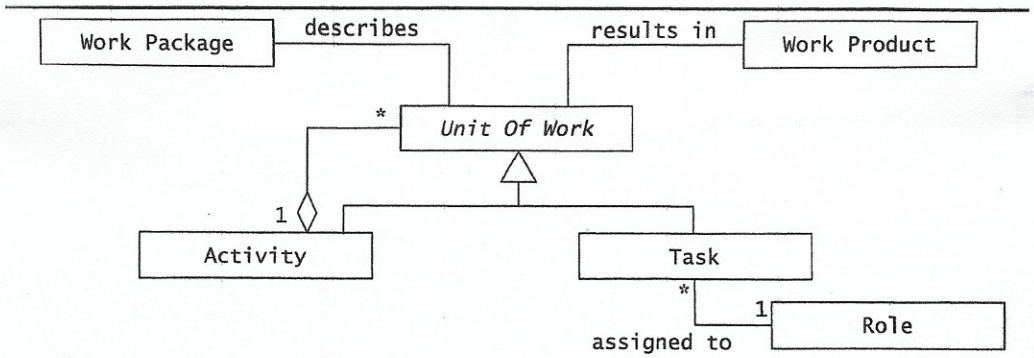

Figure 3-9 Associations among tasks, activities, roles, work products, and work packages (UML class diagram).

Table 3-3 Examples of tasks for the realization of the database subsystem.

Task name	Assigned role	Task description	Input	Output
Database subsystem requirements elicitation	System architect	Elicits requirements from subsystem teams about their storage needs, including persistent objects, their attributes, and relationships	Team liaisons	Database subsystem API, persistent object analysis model (UML class diagram)
Database subsystem design	Object designer	Designs the database subsystem, including the possible selection of a commercial product	Subsystem API	Database subsystem design (UML diagram)
Database subsystem implementation	Implementor	Implements the database subsystem	Subsystem design	Source code
Database subsystem inspection	Implementor, Tester, Object designer	Conducts a code inspection of the database subsystem	Subsystem source code	List of defects
Database subsystem test plan	Tester	Develops a test suite for the database subsystem	Subsystem API, subsystem source code	Tests and test plan
Database subsystem test	Tester	Executes the test suite for the database subsystem	Subsystem, test plan	Test results, list of defects

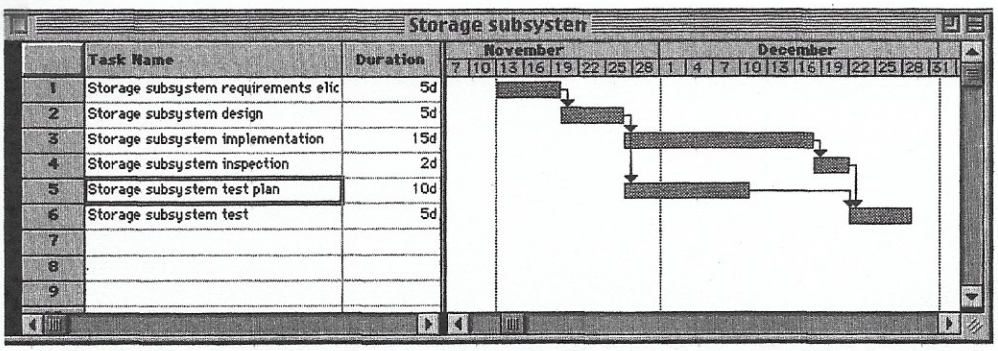

Figure 3-10 An example of schedule for the database subsystem (Gantt chart).