15 Managing object-oriented software engineering

15.1 Introduction

To introduce a new development process into an organization is seldom painless. To get all the people involved to accept all the ideas in the new process involves a lot of work from pedagogical methodologists.

It is thus essential to get the development staff adopting an organized way of working and thinking, and this as fast and painlessly as possible. No one will accept any delays caused by a new process being introduced. On the whole it is the standard problem of getting an old organization to adopt a new way of working. The organization is often, at least initially, a specific project. That is often the best form for system development.

In this chapter we will discuss some of the experiences we have had introducing and working with Objectory in real projects during the last few years. To date (Summer of 1991) we have used Objectory in about 15 projects of varying size (3–50 man years). We will first discuss some of the preparations necessary for introducing a new process and then performing projects.

15.2 Project selection and preparation

15.2.1 Introducing a new development process

Most organizations we have worked with that have chosen to use a new development process already have a significant method maturity. Most have worked with one or several development methods earlier, and they have also developed a sound skepticism against introducing new technology or new ways of working too fast.

Five levels of process maturity of a software development organization have been defined by people at the Software Engineering

Institute, see Humphrey (1989) or Yourdon (1990). This classification is gaining wide acceptance. The levels are as follows.

(1) **Initial level** No documented method is being used. Every software developer is doing it his or her own way.

(2) **Repeatable level** A method exists, but has not been formalized or written down However, there exists a consensus of 'the way we do things around here'. Often this level is reached by long experience of system development. However, when new methods and tools are introduced, the organization can very well be thrown back to level (1).

(3) **Defined level** A formal, documented process of developing systems exists The process is continuously refined by a software process group.

(4) **Managed level** Formal measurements of different characteristics of process and product are continuously performed. Not only time and cost are measured, but also productivity, effectivity, quality and so on.

(5) **Optimizing level** The measurements from level (4) are systematically used as feedback to optimize the process.

There also seems to be general agreement that an organization is not ready to adopt new methods or tools effectively unless it is at or above level (3). Investigations in the late 1980s show that 85% of the large software development organizations in the USA are still at level (1), 10–12% are at level (2) and only about 3% were found at level (3). In 1990 no organizations were found at levels (4) or (5) in the USA.

These observations from the USA perspective are very interesting. However, from our (European?) perspective, they seem a bit odd. Method maturity, we believe, seems greater in Europe. In practice, all of the projects that we have been involved in are with organizations at levels (2) or (3). About half of the projects have been in organizations at level (3), (which is said to be only 3% of the large USA data processing organizations late 1980). Our observations are also that certain organizations at level (2) have been matured to adopt the new technology. An especially important observation that we have made is that there can be a significant difference between the organization maturity and the maturity of the individuals.

Hence a big effort and much preparation are needed when introducing a new way of working. The new way means that the organization needs retraining and needs to develop new routines. This will, in a period of transition, give a lower productivity. If an organization has maintenance responsibility for several systems

which have evolved during their life cycle, a consideration of each and every system should be made as to whether it really pays to change technology for that particular system. With a matured system, which requires limited resources for maintenance, and that in the near future will be replaced by a new system, there is probably no reason for changing the maintenance strategy.

A practical way is often to introduce the new way of working stepwise. A suitable way is to select a smaller project for new development or a limited re-engineering of an existing system. We will talk more of how to select the right project in the next section.

There are certain factors that increase the possibility of making the transistion to the new way of working successful.

(1) The selection of a new development method is a very important strategic decision which must be supported by upper management. The entire organization should be aware of the importance of the decision

(2) The first development project using the new method will be exposed to much attention. Thus there is a great need for success. There will always be critics who will exploit every sign of failure to discredit the new method. The first project must thus be selected with much care and must have all the attention and resources needed to guarantee an appropriate result.

(3) The people working in the selected project must have a feeling of a positive change. This requires, for instance that they must have sufficient training that they feel comfortable with the new situation. Further, they should have tools (e.g. CASE support) which stimulate the new way of working.

(4) Introduce the method prior to any CASE tool supporting the method. Method and tool are distinct, but both are desirable. You can use a method without a tool, but not a tool without a method. Far too many people have a hard time in realizing the difference between methods and tools.

(5) The new way of working must be integrated with other routines, for example project and product management. This integration should be ready when the new order of working is introduced widely.

(6) Have reasonable expectations of the first project. It will take some years to reach a significant increase in productivity, but increase in quality will usually come from the start. OOSE will normally not be profitable in one project; the first project will be more expensive than traditional technologies, but the quality

will be better. The profits will come in subsequent projects with experienced people and also in maintenance of the first system.

(7) Do not have high expectations of components and reuse initially. The benefits will come in two or three years.

15.2.2 Selecting the first project

The first project using a new working method is often an evaluation project. It is not only a matter of evaluating the new method as such – there may be well documented experiences from other organizations – it is also a matter of evaluating the method used in this particular organization. This involves examining how inclined the staff are to changing their working method and what is involved in doing it effectively. Therefore how the first project is selected and which staff should be involved in it are essential considerations.

We suppose that the overall decisions are already made. By this we mean that upper management believes that the selection of method is a crucial and a strategic decision which they are ready to support both financially and with careful attention. There should be a person in the upper management who has a special interest in following and supporting the project. This person we call a **sponsor.** He or she should have a good professional reputation and be respected in the organization.

When selecting the first project there are certain things to consider. Generally, it should have as optimal conditions as possible for the project, to evaluate the method without any unneeded disturbances such as from shortage of staff or problems in defining the system responsibility. We summarize this in the following recommendations.

(1) Select a real project that is important, but not with a tight time schedule or any other hard constraints.

(2) Select a problem domain that is well known and well defined.

(3) Select people with experience from system development who have a positive view of changes. The management should have confidence in them.

(4) Select a project manager with a high degree of interest in the task

(5) The staff should work full time within the project and not be disturbed by other projects.

Table 15.1 The need for education when introducing a new development process: × = necessary, + = preferably

role / issue	Project manager	Analyst	Constructor	Tester	QA	Upper management
Concepts	×	×	×	+	×	+
Project management	×					
Overview	×	+	+	+	×	+
Analysis	+	×	×	+	+	
Construction	+	+	×	+	+	
Testing	+		+	×	+	

(6) Base your work on a detailed plan developed in advance. Perform evaluation at all stages with criteria established in advance.

15.2.3 Education and training

All personnel involved in the new order of work need education and training. When a method and process have been strictly defined, more emphasis can be put on formal education and training. Therefore more material can be taught, not needing several projects to learn working in an orderly way. In a less formalized method, the staff need several projects as learning projects before getting familiar and highly productive with the method.

The scope and amount of education needed varies depending on each person's role in the project. Everyone should have a basic education. The purpose is to get a common basis of concepts and way of working. Additionally, more specialized education and training is needed.

In the Table 15.1 we have summarized our experiences from the need of education.

An example of contents of the courses is given below.

- Concepts – Basic concepts of object-orientation and the fundamental concepts of the method. Chapters 3–6 in this book may form a basis. 1–2 days.
- Project management – Specific characteristics for the new method.

– Appropriate metrics for the management within the new method 1 day.

- Overview – An overview of the method using examples to get familiar and acquainted with the entire method in the system life cycle and the underlying ideas. This entire book may form a basis for this course Attendees should be familiar with object-orientation. 3–4 days
- Analysis – A thorough and detailed study of the entire analysis process. Emphasis should be on applying the method and process on a larger example, possibly on the current system. 3–6 days
- Construction – A thorough and detailed study of the entire design and implementation process. Emphasis should be on applying the method and process on a larger example, preferably the system to be built. The course does not cover the specific language, but rather how the language is incorporated within the process Those attending should be familiar with the programming language used. 3–6 days.
- Testing – A thorough and detailed study of the testing activities and principles. 2–3 days.

If CASE tools are to be used in the project, additional training is needed. Our experience is that the CASE tool should be well-delimited from the process and method issues. The attenders will otherwise have difficulties in differing in method, process and tool and too much time is spent on getting acquainted with the practices of the tool.

Any other technical aids that are new and should be used may also need additional education and training. Examples are the programming language, DBMSs and operating systems. Other examples of new areas that also require education and training are component management and usage and quality assurance. These other areas should also be supported by education and training.

Besides these courses we also need training to get familiar with the new technology. This can be done by developing a part of the system or introducing new people for simpler tasks in projects with experienced developers. All together about three months should be allocated for a developer to become productive in the new environment. Hence, it is quite expensive to introduce a new development process, but it normally pays off in a few years.

15.2.4 Risk analysis

Introducing new technologies always brings potential risks. The number of new technologies introduced simultaneously increases risks exponentially The introduction of a new development process, often involves other changes as well; a new point of view – object-orientation; a new programming language – OOP; a change of documentation techniques; new CASE tools, new development environments, new QA techniques and so on. Not only does the technology introduce risks, but the maturity of the developing organization could also be a large risk. Are we mature enough to treat system development as an industrial process? Can we introduce the discipline needed? All of these changes at the same time do involve a large risk. It is therefore essential to be aware of these risks and to have a way of handling them. To have progress one must take risks, and awareness of the risks is a prerequisite to manage them.

We will here give a simple technique for how to detect and manage risks in introducing OOSE. The method is divided into three steps.

(1) Risk identification
(2) Risk valuation
(3) Managing the risks.

For **risk identification**, we define the potential and foreseeable risks of the project that, if they occur, may seriously injure the project. The starting point should be the goal of the project. What risks may occur that will make us fail to reach the goal? These risk areas could involve OOSE specific risks, but also other risks. Examples of risk areas involved in OOSE projects are:

- *Paradigm shift*
  - Are we mature enough to adapt a new development strategy?
  - Will the team be mature enough to not start coding too early?
  - Is the project manager familiar with the new paradigm?
  - Are the team members familiar with the new paradigm?
  - Are we familiar with the programming language to be used?
- *Process*
  - Is our process well defined and well documented, and can we work along it?
  - Is the process mature?
  - Does the documentation produced fit our purpose?
  - Do we have sufficient training capabilities?

- *Tools* (analysis, design, implementation, DBMSs, configuration tools, purchased component libraries etc.)
  - Are we familiar with the tools to be used? With the learning threshold?
  - Are the tools to be used mature and stable?
  - Will we have the tools when they are needed?
  - Are the tools compatible? Are they integrated?
- *The system*
  - Are we familiar with the application domain?
  - Are the requirements clear, consistent and stable?
  - Can or must we integrate existing systems? Is it possible?
- *Organization*
  - Do we have a tight schedule?
  - Will time-to-market be critical? Will the pressure from the market change?
  - Does the organization have realistic expectations for the project?

These are just a few examples of potential risks. There are of course many more, and the risks must be identified from each project's circumstances. About 15–20 risk factors could be identified in a normal sized project (5–15 persons). Risk identification is best done on a group basis, and it is better to identify too many risks than too few. The risks should be broken down to concrete situations or events.

The next step is **risk valuation**. Now the potential risks should be evaluated to assess the consequences (disturbance or damage) if they occur. We then initially judge the *probability* that the risk will occur. A scale from 1 (very improbable) to 5 (very probable) could be used. Then we assess the *consequences* of each risk. A scale from 1 (neglegible) to 5 (catastrophic) could be used. Now multiply the probability and the consequence factor for each risk, see Table 15.2.

We will now have a relative measure of the size and potential threats of the risks.

The third step is **managing the risks**. From these measures we establish a prioritized table of the most serious threats to the project. For each of these serious threats we propose active actions to prevent them from occurring or to reduce their consequences. This could be done by decreasing the probability of the risk occurring or decreasing the consequences, or both. In this example we see that the major threats are that the project team are not used to OOSE and we are not sure about the delivery of the DBMSs. The measures to avoid these threats could be to issue an education program for the team members and not to use the DBMS proposed.

Table 15.2 A table for risk valuation

Risk	Probability to occur (P)	Consequence (C)	P*C
Development tools not mature	3	4	12
Not familiar with application domain	2	4	8
Project team not used to OOSE	5	5	**25**
Weak support from upper management	1	4	4
Delayed delivery of stable DBMS	4	5	**20**
. .	.	. .	. .

Since some risks may be hard to detect initially, we should identify warning signals that we should look for during the project. We could also initially issue alternative plans for if any of the risks should occur.

Risk analysis should be done initially in the project, but it is important to do it also throughout the project, since new risks may show up or their probability/consequence may vary during the project.

Introducing OOSE in an organization involves risks. It normally also involves a large paradigm shift with far-reaching consequences It is important to be aware of this, and, as stated, not have too high expectations of the first project, especially not in the reuse issue. As previously mentioned, OOSE stimulates reuse, but to be able to reuse, we need something to reuse. Depending on existing reusable software, the first project will normally have varying levels of reuse. However, the first project will usually yield reusable software for future projects.

This simple method of risk analysis could be used if no other method exists in the organization. The important thing is to think about potential risks prior to the project and to prepare activities to manage potential problems in the project. Please note that we cannot compare projects with this method. Neither should we mechanically conclude that any project with high risk threats should be stopped. As stated, preparing mentally for potential risks enforces the project. A more advanced method for risk analysis is the **Lichtenberg** method, see Glahn and Meland (1985).

Since a project introducing a new working method involves large risks, a risk analysis often signals red. To balance such a risk

analysis, a **benefit analysis** could also be made. What are the advantages of introducing this new working method? We will not go into details here, but just mention some major benefits of OOSE. Firstly, typically more than 50% of total development time involves information search and retrieval and coordination of activities, thus not directly efficient development time. An ordered way of working, introducing a defined method and tools, typically decreases the information search and retrieval time. Additionally, a well-defined process decreases the coordination time overhead. Hence more time can be spent on actual development. Secondly, since OOSE models the way people think, the models are much easier to understand by non-OOSE practitioners than are models produced by other classes of methods. Thus managers and customers can take a more active part in the actual development. These are just two benfits of OOSE, and there are many more like higher quality and decreasing future maintenance costs.

15.3 Product development organization

When developing a product, the basic organization of a project should be built around the product and the activities associated with the development of the product. Since the product (one hopes) will be developed in several versions, we must have the **product life cycle** in mind when discussing this topic. Here we see the important difference between method and process as discussed in Chapter 1. Thus let us discuss briefly the processes that exist to develop a product.

Product development with OOSE is built around different models that are developed in sequence. All these models must be maintained during the entire product life cycle; every further development should be accomplished by modifying these models. This means that all changes in the requirements specification should first be analyzed in the requirements model and inserted there. These should lead to modification in the analysis model and later in the design, implementation and testing. Hence all of these models must be up to date at all times during the product's life cycle. In these models, all objects will be documented on their own. The whole model will hold all of these documents together. We have earlier discussed the importance of the traceability between these models and the documents. It is important to be able to do this continuous revision smoothly.

The first model to be developed in OOSE is the requirement model. This model is developed by the activities shown in Figure 15.1.

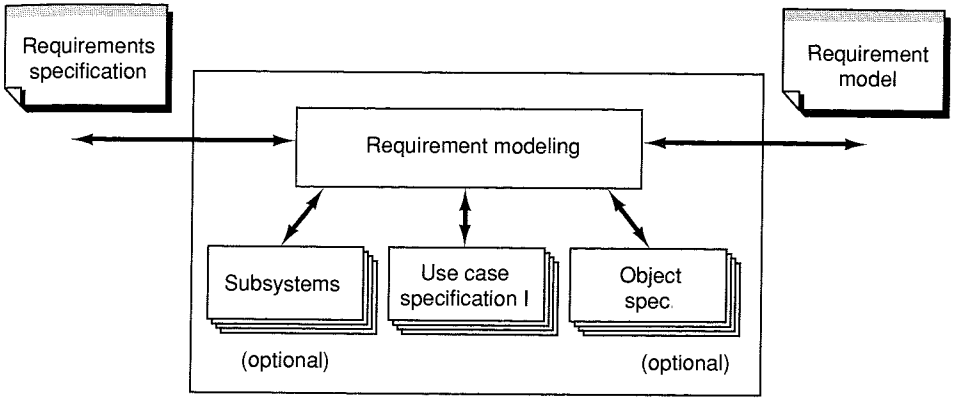

Figure 15.1 The processes of requirements analysis.

Here we have one coordinating process that lays the foundation and one process for each use case that specifies this use case. The identification of domain objects, actors and use cases is done in the coordination process while the specification of the use cases, objects and subsystems and their interfaces is done in the specification processes.

The requirements analysis process delivers a well-defined result; the requirement model with the use case specifications. This forms the input to the object modeling for the Analysis process, see Figure 15.2. The main process here coordinates three different kinds of activities: the identification of analysis objects from the use cases; the specification of each object; and the specification of each subsystem. All of these subprocesses have one subprocess instance for every such specific activity.

The analysis model, forms the well-defined result after this process. This forms the input to the construction process, see Figure 15.3. Here it is the system construction process that is the coordinating process. It has three classes of subprocesses. The first is the design of the use cases, where each use case is designed over the blocks. These activities will result in the interfaces of the blocks. The second class of process is the block construction process where each block is designed, implemented and unit tested. The third is an optional subsystem construction process for a top-down approach. Note that, as you will have noticed, we keep the design and the implementation encapsulated in the block construction process.

The well-defined result delivered by the construction process is the design model and the source code for the unit tested blocks.

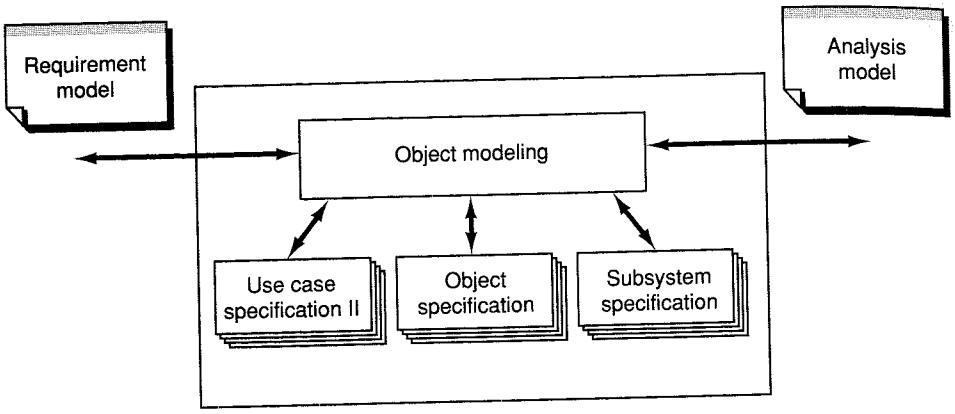

Figure 15.2 The processes of robustness analysis.

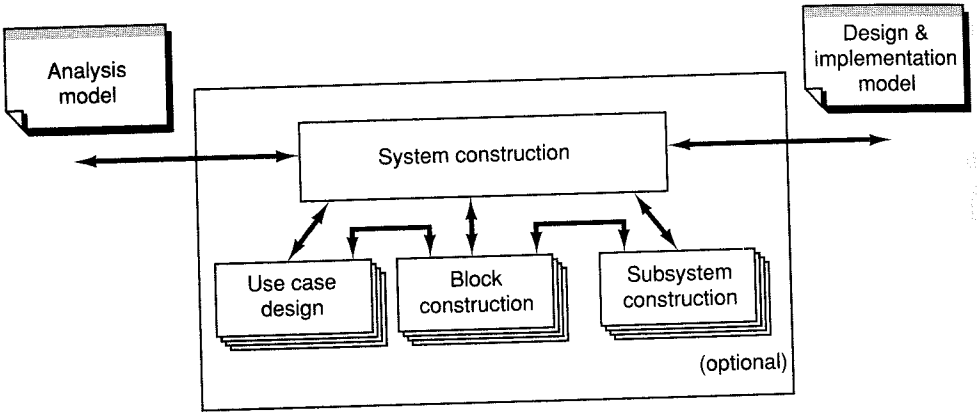

Figure 15.3 The process of construction.

This result is the input, together with the requirement model, to the testing process, see Figure 15.4. In the testing process the system is integration tested and system block tested. Here the coordinating process is the system test itself. The integration testing is performed by two subprocesses, use case tests and subsystem tests.

In addition to these three main processes we also have the component process. This process is normally not coupled to any specific product, but is a multiproduct process, that is, it is shared by several product processes. It is mainly the construction processes that it interacts with. The component process with its subprocesses is shown in Figure 15.5. The coordination process here is the

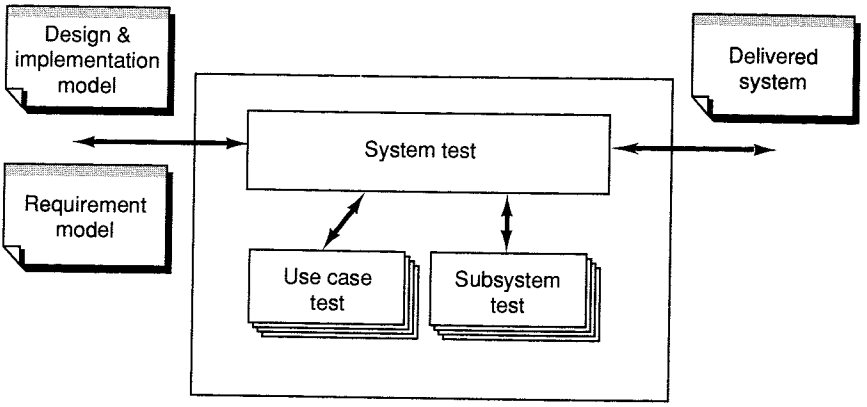

Figure 15.4 The testing process.

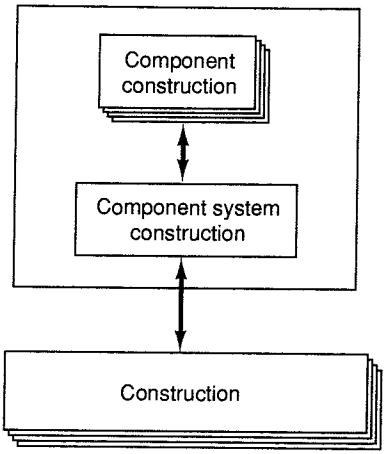

Figure 15.5 The component process interacts with several construction processes, one per product being developed.

component system construction process and the actual design, implementation and unit testing are done in a component construction process one for each component.

We have now discussed the main activities as processes in the product life cycle. The processes start when the development of the first product version starts. They then last as long as the product is maintained. Note that these processes will be unmanned when no development is being done on the product, but when a further development is to be performed the processes will be manned again.

The processes may also be concurrent activities. In this fashion we can define responsibilities for these processes in terms of, for instance, documents to be maintained Additionally, all work that is being done on a product development can be associated with one of these processes. Note here that we have generalized the development to interacting, durable processes and that we will have very similar behavior in new development as in further development. These processes do not only participate in pure development projects, but also in tendering projects, error handling projects etc

The **project organization** problem is now reduced to the question of manning these processes. Note also that the processes are similar during the entire product life cycle, that is, each new project for changing the product also uses these processes. The first version of the product will thus have the responsibility for initiating the processes, while subsequent projects just activate them by manning them. From these processes we can also identify roles played by different people during the development, and we shall discuss the importance of this later. Another benefit of this process model is that it is easier to express different kinds of development. Incremental development, which we shall discuss more later, is a very sound and common strategy, and is easily expressed in this process model. Also, the iterative nature of development is supported by the process model. Let us now turn to the discussion of allocating development staff members to these processes, that is, the actual project organization and management

15.4 Project organization and management

Managing and organizing any software development project requires a thorough understanding of the pitfalls in the trade. A necessary, but not sufficient condition for sucessful software development is good **project management**. There is an abundance of literature about how to manage software development projects. This is however not the subject of the current book and we therefore refer the reader who wants a comprehensive treatment to any of the standard works, such as Metzger (1981). Instead we will concentrate on how to combine common project management practice with OOSE and how to organize the project with respect to the product processes discussed in the previous section We will also consider what implication object-orientation and OOSE will have on project management and what project management should especially consider when working with this new technology.

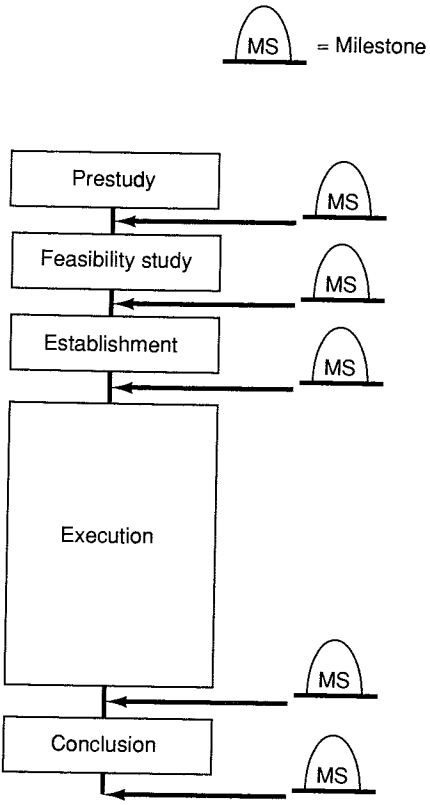

Figure 15.6 The general management part of a project

All projects will have a technical aspect and a management aspect. The purpose of the management aspect is to control, steer and follow up the project. The technical aspect covers what and how you should work to develop the current system or product. It is here that we will find the process model discussed in Section 15.3.

The management and technical aspects of a project must, however, fit together and it is common to achieve this by a number of **milestones** that should be achieved. A milestone is a concrete, objectively defined or determinable event or precisely defined deliverable. The milestones are often combined with **reviews** and **audits** of the work done so far. Between these milestones the work is performed. This division aims to give better control of the project.

In Figure 15.6 is shown a typical project model for the overall management of a project. Each specific phase is delimited by a well defined milestone.

We will here give a short overview of each phase:

- **Prestudy**. Aims at defining the task by developing and evaluating different kinds of requirements, needs and ideas to judge, technically and economically, whether the project is practicable.
- **Feasibility study**. Different technical alternatives and their consequences are investigated. A main time and resource schedule is planned and also an evaluation of potential risks in the project.
- **Establishment**. The project is organized, planned and quality assured. Detailed time and resource plans are developed.
- **Execution**. The project as such is executed in accordance with the plans previously developed.
- **Conclusion** The project is settled and proposals to improve the project and development methods used are summarized.

To this project model we should add the technical aspects of the project These include what to do in the specific phases. OOSE will be used mainly during the execution phase, although it is also possible to run the prestudy as a surveyable execution and thus also use OOSE there. However, work will also be done in the other phases that are part of the technical aspect. One example of this is **prototyping**. It is very important to achieve an understanding early of the system to be developed. The first two phases typically use prototypes. During the prestudy the purpose is mainly to evaluate technical aspects of the system to be built, often in the form of simulating certain critical parts. During the feasibility study, the prototyping is often more focused in its purpose to investigate certain technical alternatives or to support the requirements specification to be written. The purpose of these prototypes is to increase the precision and quality of the requirements, not to skip or short circuit later phases. Often prototypes can result in new prototypes to investigate certain tasks further. This is shown schematically in Figure 15.7

The prototyping technique may also be used during later phases to improve the quality of the system. It is extremely important to be aware of the purpose of the prototype; should the prototype be further refined to the system or does it just aim at investigating certain questions? Both aims are good, but far too often a good experimental prototype 'becomes' the real product, and is not what was aimed for originally Prototyping should aim at increasing the quality of the product; not decrease it.

The models developed in OOSE are good for supporting the steering of the project. All models have a well-defined result and it

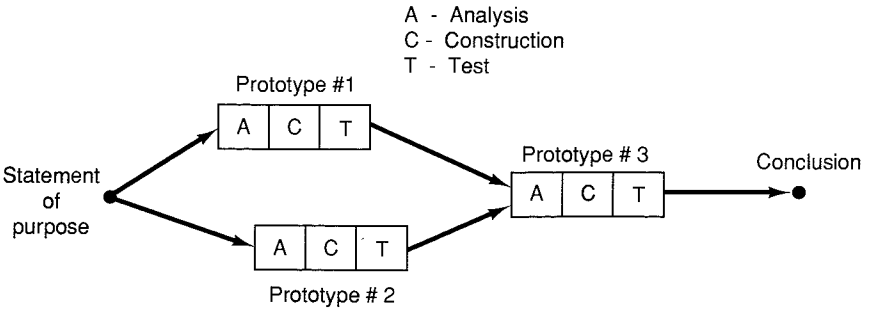

Figure 15.7 Prototypes are often an effective way of testing the validity of product ideas and preliminary system requirements They may very well result in new prototypes to refine the ideas

is appropriate to use them in combination with milestones Hence the models can be mapped onto the project model. An ideal mapping of the models is shown in Figure 15.8.

This ideal mapping gives a sense of an early waterfall model where the entire analysis model should be developed, reviewed and frozen before starting work on the design model. It is essential to realize, though, that the models will be modified when work is started on subsequent models. It is therefore essential to understand that there is no point in thinking that there will be no changes, rather it is important to have a way to handle these changes. A model which is not updated will be out of date and thus not show a consistent picture of the system. An appropriate way often is to give each model a new version at the different milestones In the following we have used an alphabetical versioning technique starting with A, B, ... Any technique could be used, we just want to illustrate the idea here. In Figure 15.9 an example of how to version the models at different milestones is shown. To handle this versioning a tool is often needed. Exemples of different kinds of documents that evolve during the project are also shown in Figure 15.9. It is also notable that the actual coding normally starts quite late in the project. We have seen that this is often an uneasy time for the team members since progress is often measured in terms of lines of code produced. Additionally, since the first project is often an evaluation project, several parties are interested in the progress of the project. However, we have also seen that when the actual coding starts, it is done quite fast, often yielding a high productivity in the overall project. Larger projects often need a more incremental development by developing the system in layers. Incremental development is further discussed in section 15.4.1.

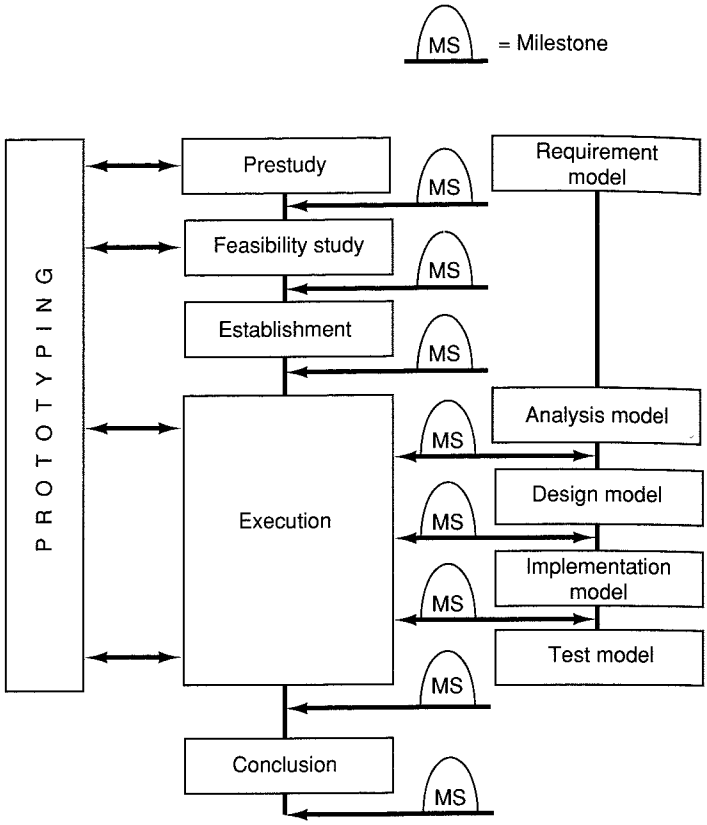

Figure 15.8 Prototyping may be used during many of the phases in the project The models in OOSE may be used as milestones that should be achieved.

The requirements specifications should be approved by both supplier and orderer and should contain a complete list of all requirements on the system. The requirements should be ranked by the orderer and the cost should be estimated by the supplier. Additional requirements may involve delivery date, resources and quality. The quality of requirements specifications varies tremendously. It is usual in technical systems to have detailed specifications while information systems more seldomly have such detailed specifications as discussed in Chapter 1. The requirements model could very well work as the final requirements specification.

Since the requirements specification is something that a contract may be written on the basis of, it may seem odd that the document will be updated and modified as the version table indicates. The hardest part when defining the requirements specification is

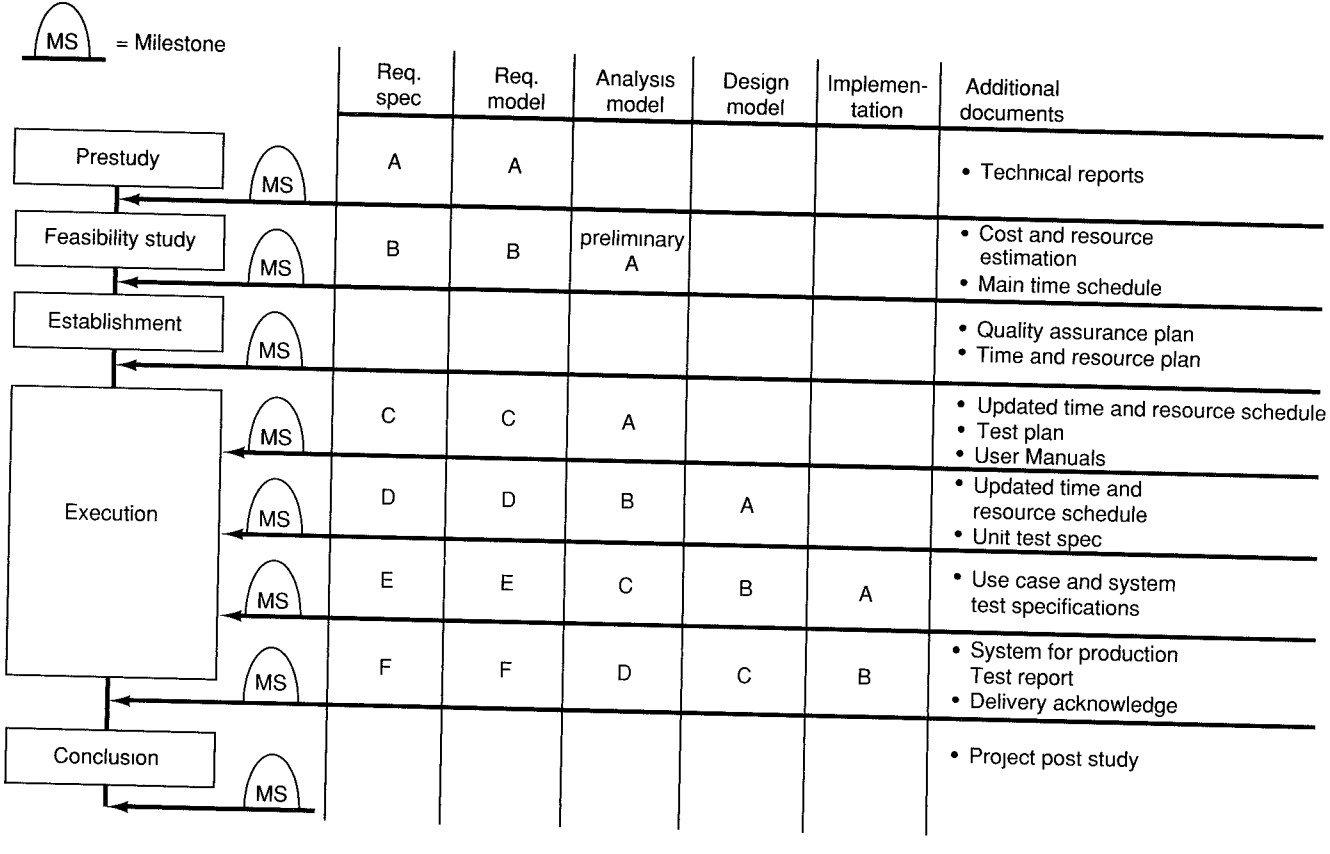

Figure 15.9 An example of how versions of the models may arrive at different milestones. Examples of documents that may be appropriate at each milestone are also shown.

often to formulate it so that the orderer and the end users can understand what will be delivered, at the same time as the developers will have a complete and well-defined support for the coming development. This document will therefore almost always be updated as the development is ongoing and hence the requirements specification (version A) that is written early on will have to be changed during the project. These changes are often aimed at eliminating uncertainties that are noticed and also incorporating additional functionality. Since the contract is of tenbased on this specification, it is important that all modifications of it are approved by both parties. This also enables the orderer to follow the development and to check that no misunderstandings occur. There are three main reasons to modify the requirements specification during later phases. The first is that the requirements are not distinct enough and need to be clarified The second is that the project notices that the time or resources available will not be enough to fulfill all requirements and thus wants to delete certain requirements. Finally, certain user groups will wake up when the system is soon to be delivered and then want to insert requirements that actually should have been in the first version of the requirements specification.

A point that is too often forgotten is that new or changed requirements will inevitably generate new costs and possibly also delay the project. Remember also that it gets more expensive to add new requirements the further you have come in the development cycle. A requirement that was estimated at 200 manhours in the specification phase, but was left out, may very well cost 1000 manhours if introduced in the design phase

When talking about timing we will only mention an obvious fact that is too often forgotten. If you delay one phase of the project, it will inevitably delay the delivery date also. The subsequent phases will not suddenly be done faster just because one phase took longer. Here we see the importance of following up the project plan and schedule.

15.4.1 Incremental development

Although it is possible to execute a development as discussed above, experience shows that an incremental approach is often more appropriate Since the models are developed in a very seamless way, it often feels natural to 'investigate' what will happen to one object in one model when continuing to the next model. Object-oriented modeling is often connected to an iterative way of working, probably

because of the strong traceability between different models; a problem domain object may exist all the way to code. However, to work too iteratively is not good either. Then you will use later models to correct earlier models; much the way compilers often are used to debug programs. The solution lies of course in between: first develop one part of one model and then continue this part to the subsequent models and then continue with the next part of the first model.

When using OOSE it is often appropriate to develop the requirement model quite extensively as a first step. One reasonable goal is at least having all use cases *identified* for the entire application. The reason for this is that it is important to have an understanding of the entire system before starting to structure it. It is then possible to start with a couple of use cases and specify them and then continue with these use cases into the analysis model and also the design, implementation and testing phases. In this way we take the use cases of highest **rank** not designed yet and refine the latter models with the new use cases. Here it is evident that the use case is the red thread through all the activities. Testing accumulates the use cases, and when all use cases have been tested, the final system test is performed. This development is illustrated in Figure 15.10.

Each increment should be of reasonable size, not too small and not too large. Increments of about 5–20 use cases are often appropriate. It is also important that each increment covers a limited time; often 3–6 months is a reasonable turnaround time and preferably not more than 12 months. We have seen in projects that project members are often unsure how much work is to be done in one model and how that work is to be used in later models. When this is the case, it is often appropriate to have a fast turnaround, to gain experience from work in all models early. Working this way, the developers will early on have a better understanding of the whole development process and can thus optimize the process as such.

In this way the different phases may also overlap For instance, when doing construction, analysis can be performed in the next increment, see Figure 15.11. Since one increment may modify results from previous phases in the same increment, it is important to have a very controlled way of handling different versions of the models. Additionally, you must be able to handle the modification and updating of models in a dynamic way, for example modifications in the analysis model in the first increment should be incorporated smoothly into the analysis model for the second increment.

It is essential to minimize the work needed to deliver results from one phase to another. The deliverables from each phase are defined by the process outputs A way to solve this is to have some developers follow one increment through all phases and other

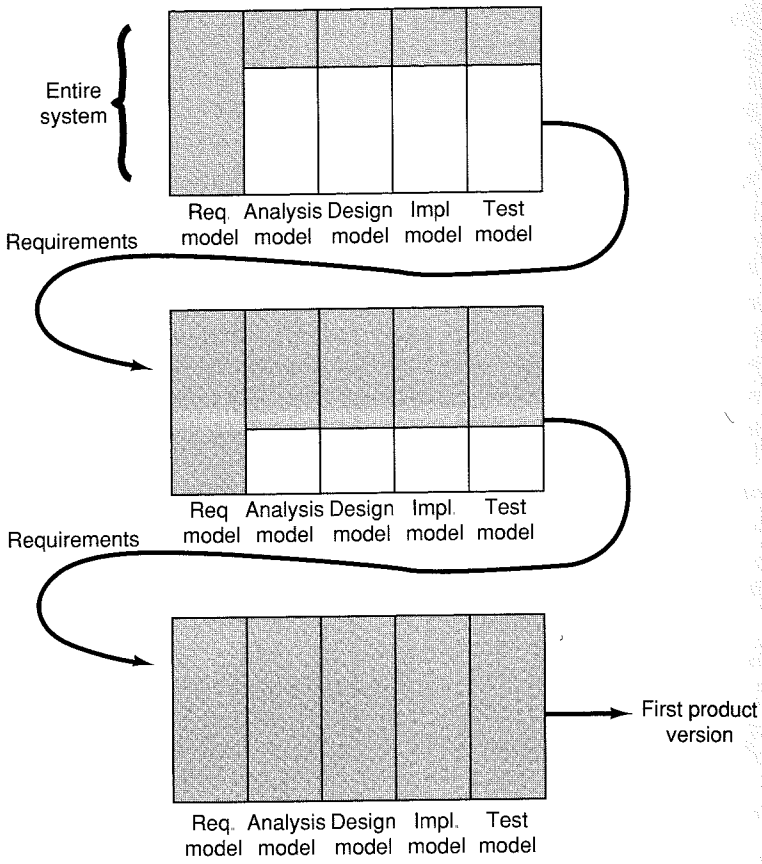

Figure 15.10 Incremental development in OOSE.

developers follow one process over several increments, that is, to have some specializing in a couple of use cases and others specializing in for instance analysis

15.5 Project staffing

The number of people involved in a project varies highly in different phases. Different phases also need different competences. Additionally, the organization needs to be changed over these phases to manage the project. This is due to the fact that in some phases work can be done in parallel while this is harder in other phases, but also since some phases require more resources than others.

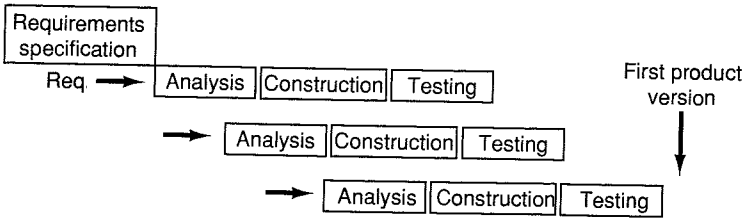

Figure 15.11 Work in different phases may very overlap Then it is essential to have support for handling different versions of the models.

Generally, it is valuable to have people participate both in analysis and construction and thus ease the transition between different models. Therefore a development core of staff can be the core of the entire development. However, we may have people specializing in analysis, construction and testing. Especially (integration and system) testing is often performed by a special group. The test specifications, though, are developed from the use case description from design and are based on which blocks are involved in the use case and when those blocks will be delivered to testing. Design and implementation should thus be planned so that the blocks can be delivered to testing when they are needed in testing. In this way much of the work can be done in parallel. This means that even if other people are involved during testing, they must collaborate intensively with the construction personnel

We will here briefly discuss some typical ways to organize a project and also illustrate some specific points in an object-oriented development. We here assume a medium sized project which involves 5–20 people during analysis, construction and testing. Staffing in other phases than the ones we have discussed in this book will not be covered.

When assigning development staff tasks in a project, we use the processes discussed earlier After all, since the processes describe what is to be done, a specific project is actually a flow over these processes. The problem is thus to man the processes for a specific project Let us start with the coordinating processes. These are summarized in Figure 15.12.

These main processes coordinate the entire development. The identification of use cases is performed in requirement analysis, while the specification of them is done in specific subprocesses for each use case specification. These main processes will structure the

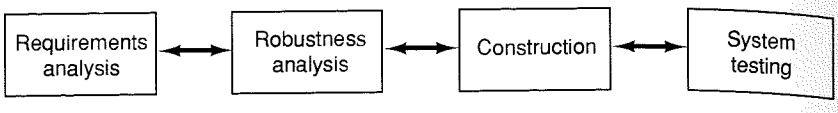

Figure 15.12 The coordination processes in OOSE.

system robustly (in robustness analysis) and make the final architecture (in construction). Hence the important structuring of the system is performed within these processes. The people staffing these processes should therefore be the core of the development and thus should be the same in at least the first three processes. We may call this group the **system architecture group**. Their responsibility is to make sure that the system architecture, and its coherent idea, is maintained during the entire development. It is essential that these people are highly qualified and have a strong influence on the project members and that the rest of the project members have confidence in these people. The project manager should be tightly coupled to this group.

The initial surveyable analysis (mainly requirement analysis) should be made by quite a small group with much interaction with the end users. All people involved in requirements analysis do not need to know the technology used in later phases; the concepts used are quite intuitive. These people should have a close contact with end users, customers, marketing people, experienced developers and so on.

The subprocesses for the more detailed work should be manned by development personnel that have special skills for the activity. Here it is often good to have the same person responsible for the same group of objects in all activities. For instance, the person specifying a specific use case should also specify the objects that offer the use case, design that use case and implement the corresponding blocks. However, since the complexity (i.e. work) increases, especially during implementation, this is not always possible. Nevertheless, the person responsible for the specification in analysis should work as a senior designer when implementing the blocks. Similarly there are often people involved in modeling the requirement model who will not participate in the subsequent work.

The main reason to have the same person in all subprocesses that manage some objects is to minimize the work of collecting and understanding information. Additionally, you will avoid the conflict of not consenting with the specification (the 'not invented here' problem). However, this may also involve drawbacks. When doing

it this way, reviews will be more important Other persons must also be able to understand the documentation for further development.

In larger projects it is often appropriate to divide the development into several project groups Each group is then responsible for one subsystem and/or a specific task in the development. It is far better to have one group responsible for one part of the system than to have one group responsible for a specific phase. The reason for this is that the domain knowledge, for example the knowledge of the subsystems task, is more important for the result than is detailed knowledge of a specific phase. In each project group, the activities are divided further. All these activities are cooperating subprocesses in the development.

When reaching construction, typically more people are involved This may be solved by adding more people to existing groups or to add new groups in the project organization. When you add new groups you will also lessen the responsibility of each group. It is often more appropriate to have smaller subsystems assigned to each group instead and to keep the initial groups. Generally you should not let the resources affect the system structure. The new people can have other knowledge profiles than the people in the analysis phase. Block testing is done during construction and thus is usually an activity within each and every project group. These phases also need a coordinating architecture group, preferably the same as during analysis.

The component activity is responsible for maintaining and developing components. This is an activity that should be shared among several projects and thus not a part of the project. It is essential that upper management realizes this important distinction.

Integration testing is done in a separate testing phase, often by a separate group. This group can very well start its activites by writing test specifications when use cases are specified during analysis. When all use case tests are performed the final activity within the development is the system test This test involves testing the functionality of the system, documentation and tutorial material. The system test cannot start before all the parts in this version have been delivered from construction.

During or after the testing phase, it is suitable to do a poststudy of the project where positive and negative experiences are collected and documented It is better to do this study directly after the project when the members are available and their picture of the project is still clear and to complement this study with a study of operation experiences of the system after about 3–6 months, than to do the whole poststudy then. The risk then is that the poststudy will never be done and hence valuable experiences are never documented.

Besides the actual developing groups there may be a need for other roles or groups in the project. Examples of this are listed below:

- **Methodologist**. A person or group responsible for the method used. He or she or they should be experts in the method and support the development team in applying the method. It is often useful for these people to understand 'both sides', namely to be able to explain the new techniques in terms of the techniques already used. This means that the methodologists must be experts, not only in the new method, but also in the language, operating system, product structure and development organization of the team where the new method should be introduced.
- **Quality assurance (QA)**. People responsible for both the product and the process to develop the product so that it is of high quality. This involves guaranteeing that all software delivered is of high quality and that documenataion is consistent. Reviews are here a useful tool and we will discuss this issue more later.
- **Documentation, manuals and education**. Documentation of the system should be made by the developers. It should of course be consistent with the system developed. Manuals, both for maintenance and for users, should be written by people with special skills for this. Planning for education on the system must also be done. People who should be trained include users, maintenance and operation people and sales staff.
- **Reuse coordinator**. This person or group is responsible both for encouraging and evaluating how much the project is reusing and also investigating the reuse potential of the code and designs developed. Code, designs, documents and models may be reused. This person or group should work intimately with the architecture and QA group. Reuse might not give a pay-back in one specific project; the real gain comes in subsequent projects that reuse what has been developed in this project. Therefore the cost of this function should not burden the project cost, but rather be considered as a multiple-project cost. Note that the coordination and management of the reuse library should be interproject, see Chapter 11.
- **Prototyper**. This role is necessary to investigate different solutions at an early stage to prepare for later development. Typically user interfaces are interesting to prototype in early phases, but simulation of certain designs in later phases may also be interesting.

- **Support environment** This is to function as a service to the project as a whole Typically system managers play this role.
- **Staff**. To help the project manager, staff may be needed. One example is a special project administrator responsible for following up cost and time schedules. This is often needed in larger projects. Some of the above roles may be part of the staff.

It is often effective to have people with these special roles also working in the project groups. If so, it is important to give them time to fulfill their double roles.

15.6 Software quality assurance

Software quality assurance (SQA) aims at ensuring that the final product will have an acceptable quality. It is mainly a management activity to identify quality problems early in the development. Cost and time schedules are often tracked in the early stages, as opposed to quality, but quality problems appearing late in development involve large risks for any project. Therefore it is just as important as tracking cost and time to track the quality.

Quality assurance focuses on both the product and the process. The **product**-oriented part of SQA (often called Software Quality Control) should strive to ensure that the software delivered has a minimum number of faults and satisfies the users' need. The **process**-oriented part (often called Software Quality Engineering) should institute and implement procedures, techniques and tools that promote the fault-free and efficient development of software products.

What then are the characteristics for high quality in a product? In Figure 15.13 some of the characteristics are shown.

These characteristics are not exhaustive and not even independent of each other. Additionally, they often tend to conflict in a development. Therefore, when starting a development, a good point is often to decide what characteristics are the most important for this specific product and then focus on these throughout the development. In OOSE the focus is on maintainability characteristics. As we have discussed, the maintenance of the product is the major objective when developing the structure of the system. However, this will of course also have effects on the suitability criteria; if it is easy to introduce changes to the system, it will also decrease the number of faults introduced when modified and thus give the product a higher reliability.

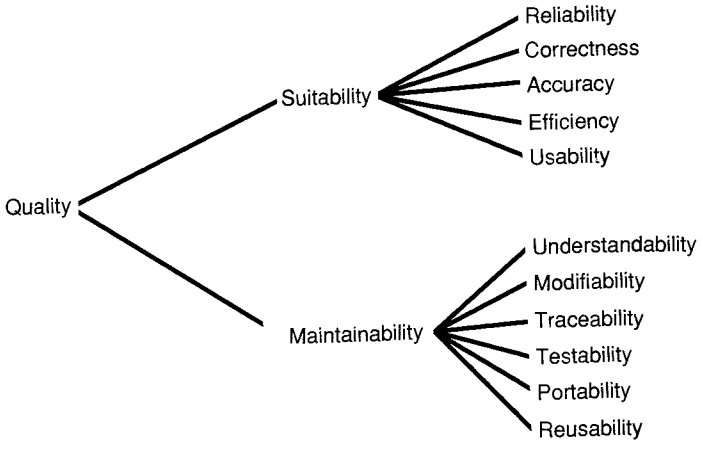

Figure 15.13 Some characteristics of software product quality

The material to work with when doing SQA is mainly the documentation produced during development. No new documents should be needed for SQA. Therefore it is essential that everything important that is done should also be documented. The (far too) common picture in software development is first to do the work and then to document it. The right way is first to document what should be done and *then* do the work. When working with OOSE this is permeated The requirement model will be developed first followed by the analysis and design models, before writing the actual code. Each of these models will be developed and documented concurrently. Therefore OOSE gives a good platform for carrying out quality assurance in an accurate way.

The main tools for quality assurance are the development process itself, reviews and audits, testing and also metrics. The development processes of OOSE, including testing, have been surveyed in this book. Metrics will be discussed in the next section. Here we will discuss very briefly the integration of reviews in OOSE. First some terminology: a **formal** review's objective is to decide whether or not to proceed to the next phase. Such a review is held at every major project milestone A quite large review team is often involved and also customers or orderers participate. An **informal** review's objective is to discover errors that have been made. These reviews can be held at any time during development, such as when something is completed that ought to be checked before continuing the development. Informal reviews often have a quite limited participation, typically some of the developers.

Where to use different kinds of reviews when working with OOSE depends on the size of the project. In a small-to-medium sized project, typical formal review points are between the main activities, that is, each model when it has reached its first version. Informal reviews may be used after each subprocess, possibly grouping some subprocesses in one review.

Different kinds of reviews have also been defined by IEEE (1983) in a standard glossary. Three different kinds of reviews are then given:

- **Review** A formal meeting at which a model is presented to the user, customer or other interested parties for comments and approval,
- **Inspection**. A formal evaluation technique in which models are examined in detail by a person or group other than the author to detect errors, violations of development standards and other problems,
- **Walkthrough**. A review process in which a developer leads one or more other members of the development team through a segment of a model that he or she has written while the other members ask questions and make comments about technique, style, possible error, violation of development standards and other problems

Of these we may characterize review as formal and inspection and walkthrough as informal. Although every review is unique and focuses on a specific model or object, they have some points in common We will here not give exhaustive lists of what to review in the OOSE models, but rather just highlight some examples.

Common to all reviews is to check things like consistency with requirements, completeness of model, redundancy, structure, naming, correct associations, understandability, versioning of documents, standards and views. This is reviewed more or less dependent on the purpose of the review

When reviewing each model there are specific things to focus on We shall here only give some examples of points to review in the Requirement model.

- Is the system delimitation appropriate?
- Do the use cases match the requirements specification?
- Have the requirements specification been updated?
- Is it possible to understand the use cases?
- Are all roles interacting with the system identified as actors?
- Do all actors have the right set of use cases?

- Has the requirements specification been covered?
- Are the interfaces described in a satisfied way?
- Are the use case flows correct and complete?
- Have enough alternative flows and error flows been described?

Of course the actual questions may vary from time to time, but the intention should be clear from the above. When reviewing the other models the intention should be the same; to find errors as early as possible in the development process and similarly to guarantee a high quality in the product.

When performing the review, different methods and techniques can be used. A systematic approach to this is a necessity to achieve a high quality of the delivered system. Methods and techniques for this are described in the literature, see Weinberg and Freedman (1982), Myers (1987) and Yourdon (1989b).

When performing reviews, management has an important role to play. It is important that the management show commitment to the process and results and also budget time for performing reviews. More than 5% of the overall development time is not unusual. Likewise it is important to get good people as reviewers to guarantee a high product quality and high confidence with the review process from the development staff. The manager should also reward good reviews of poor products and punish bad reviews of any products. There will always be errors to detect. Some hard facts about defects are the following:

- A defect introduced during requirements specifiation will cost 100–1000 times more to correct when the system is in the testing phase than it would cost to fix it during requirements specification.
- Between different programmers the number of faults introduced may differ by as much as a factor of 10 when producing 1000 lines of code
- During normal testing only about 50% of the faults will be detected.

To achieve a good quality discipline, and a high quality awareness, an independent quality group responsible for quality assurance in the development department may be needed. This group can do reviews of how well the project members follow the given process of development and can also make an assessment of the project's possibilities of achieving its goals and illustrate potential risks. However, the QA group should not function as policemen, but

rather, together with the development team, increase the quality of what is being done

Finally, we want to give some quality advice:

- Follow the development process thoroughly, and note what goes wrong,
- Eliminate the faults as quickly as possible by revieweing all specifications thoroughly,
- See that the review groups have the right composition of people,
- Note in the review protocols the number of pages reviewed and the number of faults of different types found,
- Follow up the review protocols and identify, and possibly rewrite, extremely errorprone objects. Try also to identify any individuals who seem to produce many defects in specifications or code,
- Have an independent testing group testing the system and also writing the test report,
- It is always cheapest to *do it right the first time.*

15.7 Software metrics

'If you can't measure it, it's not engineering'.

A necessary way of controlling a development is to use metrics. The metrics can measure either the process of development or measure various aspects of the product. Metrics in software engineering have been discussed for a long time, but not used widely as a way to increase quality of the product or process The real problem is that we cannot measure exactly what we would like to measure; we must assume that there are relations between what we can measure and what we would like to measure. Process-related metrics have been used much more than product-related metrics, so let's start by discussing this.

Process-related metrics measure things like manmonths, schedule time and number of faults found during testing. To learn to handle and manage a development process such as OOSE it is important to start collecting data on these measures as methodically as possible Below are examples of a number of process-related metrics that it is proposed to collect when working with OOSE.

- Total development time,
- Development time in each process and subprocess,

- Time spent to modify models from previous processes,
- Time spent in all kinds of subprocesses, such as use case specification, object specification, use case design, block design, block testing and use case testing for each particular object,
- Number of different kinds of faults found during reviews,
- Number of change proposals on previous models,
- Cost for quality assurance,
- Cost for introducing new development process and tools.

These measures may form a basis for future planning of development projects. For instance, if we know the average time to specify a use case, we can then predict the time to specify all use cases when we know the number to be specified. Statistical measures (such as averages) should always be accompanied by the certainty of the measures (such as the standard deviation). Otherwise you will have no sense of the accuracy of the prediction. We have also noted that these measures may vary greatly between different projects, organization, application and staffing. Therefore it is dangerous to draw general conclusions on existing data without looking at the circumstances. For instance, in one project a typical complete use case design including all alternative courses and error courses could take about five days. In another project where the use cases are smaller a typical use case design may only take two days.

For **product-related metrics,** several different kinds have been proposed. None of these have been demonstrated to be generally useful as overall quality predictor. However, as we discussed in the previous section, some quality criteria can be used to predict a certain quality property. One example to measure traceability is to measure how many of the original requirements are directly traceable to the use case model.

Traditional metrics on products (including code) may to some extent be used also in object-oriented software. However the most common metric, lines of code, is actually even less interesting to measure for object-oriented software. The less code you have written the more you have reused and that often (but not always!) gives your product a higher quality. To get a feeling of the actual code the following are examples of metrics that are more appropriate for object-oriented software

- Total number of classes,
- Number of classes reused or newly developed,
- Total number of operations,
- Number of operations reused or newly developed,

- Total number of stimuli sent.

Some metrics that are more specific are:

- Number, width and height of the inheritance hierarchies,
- Number of classes inheriting (or using) a specific operation,
- Number of classes that a specific class is dependent on,
- Number of classes that are dependent on a specific class,
- Number of direct users of a class or operation (the highest scored are candidates for components).

It is often also interesting to measure more statistical metrics. When this is the case you should measure both the average and the deviation. Some examples of such metrics are:

- Average number of operations in class,
- Length of operations (in statements),
- Stimuli sent from each operation,
- Average number of descendants for a class,
- Average number of inherited operations.

When measuring code it is important that things like comments do not disturb the metric. Therefore it is often appropriate to differentiate between lines of code, lines of comments and lines of documented code (the sum of the two previous items).

Other common source code metrics can be used with various degrees of usefulness in object-oriented software. For instance, the **McCabe cyclomatic complexity**, see McCabe (1976), measures the complexity of a graph, see Figure 15.14. The idea is to draw the sequence a program may take as a graph with all possible paths. The complexity, calculated as connections−nodes+2, will give you a number denoting how complex your program (sequence) is. Since complexity will increase the possibility of errors, a too high McCabe number should be avoided. Some standards require that no module should have a higher McCabe number than 10. Note also that the McCabe number also gives you the number of test cases to do path testing.

In object-oriented software the McCabe number as defined above will be of less interest. There are several reasons for this. The first is due to the polymorphism. As we noticed in Chapter 12, every stimulus sent is a potential CASE statement in a procedural language. Since we do not know about the receivers class, any stimulus statement could hide a various number of operations – in an untyped language, in principle as many as there are classes in the system.

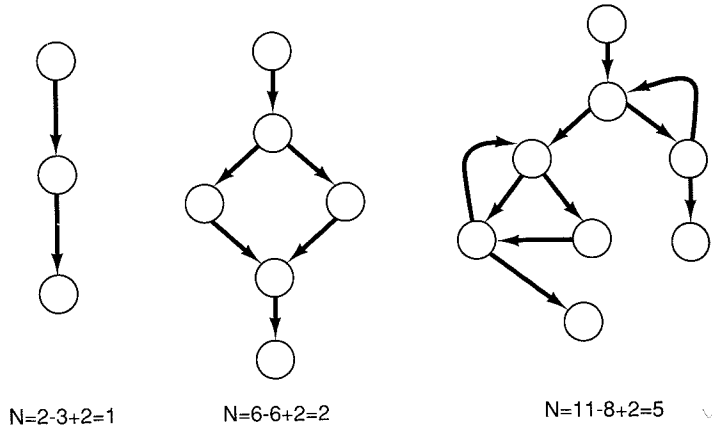

Figure 15.14 The McCabe complexity metric. N = Connection—Nodes +2.

Typically CASE statements increase the McCabe number rapidly. Thus we must decide how to handle polymorphism when using the McCabe metric; in its traditional application it is usually not very interesting

Another issue is that you would only measure the complexity of an operation since that is where you have a program sequence. It would be very unusual to have operations with a higher McCabe number, and definitely not over 10 (unless in very special circumstances), if we do not count polymorphic statements.

However, the McCabe number could be used in OOSE. The use cases connect together several objects in a specific sequence. This sequence will have a complexity that is of great interest. To calculate the McCabe number for a use case gives a complexity measure of that use case. Here the interaction diagrams are used as a tool to calculate the metric.

We have thus far mainly discussed metrics of code. What is often more interesting (and much harder) is to develop metrics to measure the quality of the design and analysis. Today we do not have any such generally applicable metrics. However, it is interesting to collect data and measures even for these models The main reason for this is for project management, where you are interested in quantitative metrics to be able to do project planning and control. Here we will give some suggestions of what to measure.

- Number of requirements,
- Number of use cases and actors,
- Numbers of objects divided on entity objects, interface objects and control objects,

- Number of subsystems,
- Number of blocks,
- Number of classes.

Besides these absolute metrics, it is also interesting to measure:

- The number of objects offering a use case,
- The correlation between analysis objects and blocks,
- The number of blocks participating in a use case,
- The number of classes in a block,
- The number of operations per block,
- The number of stimuli sent in one use case,
- The number of parameters in every stimulus,
- The locality of requirements expressed for subsystems and/or use cases

The metrics that we have discussed above should not be viewed as the only interesting measures to be taken, quite the opposite, they are only proposed metrics and should inspire you to develop your own metrics. We are at the moment not ready to give clear recommendations as to which metrics should be used when working with OOSE for quality assurance and project management.

The number of different kinds of metrics is very large, and which to choose must be decided from time to time. It is important, though, to use metrics and collect data in an organized way Actually the real problem with metrics is that they are not used. And since they are not used we will not collect any data and thus cannot validate or calibrate the metrics and therefore we do not have any really useful metrics To start to break this vicious circle we must start to collect metrics and then refine the metrics as we learn about them. However, even if we have metrics it will not be the final answer. To quote Albert Einstein: 'Not everything that counts can be counted and not everything that can be counted counts ' More on software metrics can be found in Boehm (1981) and Grady and Caswell (1987).

15.8 Summary

The introduction of a new development process into an organization must be done with great care. Such change are normally part of a long term plan for the development organization. It is essential to

introduce the new process smoothly, and often a pilot project is selected to try the new process in the organization. To give the organization and the project a fair chance, a special education and training program should be planned. Risk analysis should also be done prior to starting the project to increase awareness of the risks involved, and also to take steps to manage these risks.

The focus of each project must be on the product developed. The subprocesses of the development process describe this handling and these are therefore appropriate to use for the organization of the project. The project organization is thus mainly a matter of manning all the subprocess instances. A project model to steer the project should be used. This should support the idea of milestones to follow up the project and to check that the project is on the right track all along. Prototyping is normally an integrated part of all development to increase the quality of the final product. It is essential to keep track of the documents produced and a versioning strategy should be coupled to the development process. A more complex, but often better strategy is to use incremental development. Then we have shorter turnaround time for each phase, and we also have the possibility to gain experience early.

Project staffing for OOSE involves some new roles, but also several roles that are also important in traditional development. It is essential to have a core group following the entire development that is responsible for the overall architecture and philosophy of the system. It is also important to have knowledge about the system in all phases. The use cases support such a common thread in all phases. It is therefore often appropriate to have team members following some use cases throughout development. Other roles in OOSE involve methodologists, reuse coordinators and prototypers. The team responsible for reusable components is shared among several projects and thus is a concern for upper management.

Software quality assurance should focus both on the development process and on the product developed. The fundamental techniques used include reviews, testing and metrics. Reviews is the best known technique today that gives the highest increase in quality early in development. Metrics should also be used to start collecting data to increase quality. However, there are today no well-known direct metrics that we can use to increase the quality in advance. The indirect metrics include the number of faults identified or the number of changes in a review. However, to gain quality in an ordered way tomorrow, we must start using metrics today.