Chapter

77 *SOMT Projects*

This chapter describes how to organize the activities and models of SOMT into a project. The description is intended to be suitable for a medium size project.

SOMT Projects

A development project that uses an object oriented approach like the SOMT method is not very different from any other development project. The project has a starting point, the project members work hard and finally the project is finished, hopefully successfully accomplishing the goals of the project in terms of quality, functionality and delivery time. The skill of planning, managing and executing a successful development effort has been studied and described in many books and it is outside of the scope of this document to give a complete treatment of this topic. A recent book that treats management and planning issues for object oriented projects is "Object Solutions" [32] which is recommended for further reading.

Nevertheless, this section is intended to give some indications on how to organize the activities and models of the SOMT method into a project. The description is intended to be suitable for a medium sized project where the effort is big enough to force the development to be divided into several development teams. A smaller project would most likely need less formalism in the development process and a bigger one more control.

Project Phases

From an external point of view (like a manager's point of view) it is useful to organize a project into a number of *phases*, where each phase represents work performed during a limited period of time resulting in a set of artifacts, deliverables that summarizes the state of the work after the phase is finished. There is a fundamental difference between the phases of a project and the activities as used in this document to describe the SOMT method:

- The phases are used to organize the calendar time of a project.
- The activities represent a categorization of different kinds of tasks by their nature, not by their relative time in a project.

Nevertheless there is a strong relation between the phases of a project and the activities that has to be performed. There is a dependency between the activities of the SOMT method that indicates an ordering in time between the activities. There is no reason to start the system/object design before a substantial amount of work has been done on the system analysis. Likewise there is no reason to start the system analysis before

there is a common understanding of the requirements elaborated in the requirements analysis. On the other hand it would be a waste of time and even dangerous for the project to get bogged down with the details of the requirements for a substantial amount of time without considering their consequences in terms of architecture and design.

The dependency between the activities indicates that a fruitful way to organize a SOMT project is to consider it as a sequence of iterations of the activities. The phases are introduced as a means to get a visibility of important milestones and deliverables from the project. One way to define the phases of a SOMT project is as follows:

- Prestudy/conceptualization phase
- First iteration
  - Requirements analysis phase
  - System analysis phase
  - System design phase
  - Design/implementation phase
- Elaboration phase
  - Planned consecutive iterations

Each of these phases has a section of its own in this chapter.

Multi-User Support

There is often a need to facilitate multi-user support when working with an SDL system managed by Telelogic Tau. That is, we want to have a development environment capable of handling parallel development. One way to get support for the development process is through using Configuration Management (CM). With CM is usually meant the check in and check out control of sources and binaries, and the ability to perform builds of these entities.

If CM is to be used within a project it has to be planned for. This planning is not something that is carried out in isolation. It is a part of the overall project planning process and is implemented throughout the life cycle of the project. The plan should cover activities to be performed, the schedule for the activities, the assigned responsibilities and the resources needed (including staff, tools and computer facilities). A key

task is to decide exactly which items that are to be controlled and define the engineers responsible for these items. During the development process the code is usually stored in a database based on a revision control system. The developers checkout a copy of it to their own personal workspace. Changes are then made locally and checked in when completed and tested. The developer is responsible for making sure that the module works before he checks in the changed files.

When several users work on an SDL system, the management of the system file may be difficult to synchronize. This problem is solved if every user or group has control over their own part of the system file. The multi-user support in Telelogic Tau is based on letting the users split the system file into several files, called *control unit files*. Where to split the system is an active action taken by the user, but the idea is to partition the system according to work responsibilities and to assign a control unit to each partition. Apart from supporting parallel development, the control unit files also facilitate merging of the individual results to one common system file. An example is a large SDL system where there are several blocks on system level developed in parallel by different developers. Each block could then be associated with a control unit file. Now, the blocks can be updated independently and the changes are shaping the local control unit files. Also, there is no need to manually merge changes into the system file when the work has been done. This is because the management of control unit files, performed by the Organizer, includes the merge.

Another way to get support for parallel development is through using the object oriented concepts of SDL, i.e. types and packages. Instead of introducing a block reference symbol for every subsystem we introduce a block type. The block types are placed in packages together with the signal and data type definitions that are needed within that particular block. If two blocks need the same signals and data types then these definitions are placed in a package of their own. Other packages can then use this package. In the same way as we define block types we can also define process types instead of process reference symbols. The different packages containing the types can be analyzed independently. We can also create instances of the types to simulate, verify and validate the behavior of a particular part of the system. When all parts of the system have been designed and tested we put them together, i.e. we create instances of the types and test and verify the overall system behavior.

Prestudy/Conceptualization Phase

The prestudy/conceptualization is the initial activity needed to establish the core requirements that satisfy a well-defined user need and to identify the risks involved in the project by establishing a proof of concept for the project. It is not in the scope of the prestudy/conceptualization to completely analyze all aspects of the requirements, it should rather establish a vision of the underlying idea. It should also validate the assumptions on user needs and implementation technology. One way to do this can be to build an early prototype that illustrates the goals of the project. The prototype may also be an aid in the identification of risks, e.g. related to technical aspects or to the functionality of the system.

In many cases the prestudy/conceptualization can be viewed as a project by itself, following essentially the same steps and phases as any other project, with the exception that the time scale is shorter and that the result is not a finished system. The tangible results from the prestudy/conceptualization should be:

- A clear vision of the requirements (possibly illustrated by an early prototype)
- A risk assessment that identifies the major obstacles ahead

Requirements Analysis Phase

After the prestudy/conceptualization phase it is time for the first iteration through the activities in the SOMT method. This will go through the requirements analysis, system analysis and system/object design and sometimes also the implementation activity. This first iteration is in some sense the most important part of the project since it involves both an analysis of the requirements on the system and the development of an architecture that will meet these (and future) requirements. If this first iteration is successful the further elaboration and refinement of the system will work without any catastrophes. If the first iteration is not successful, e.g. an insufficient understanding of the requirements leading to an inappropriate system architecture, there are loads of problems ahead.

The requirements analysis phase starts the first iteration. The activities of this phase are of course centered around the analysis of the requirements as detailed in chapter 71, *Requirements Analysis*. When this phase is finished the analysis team must have established a common vo-

cabulary and a common understanding of the system's desired behavior. The deliverables are first versions of the models associated with the requirements analysis:

- The textual requirements
- The data dictionary
- The requirements object model, including both information models of the problem and context diagrams
- The use case model
- The system operations model

The most important of these are the use case model, the requirements object model, including the context diagrams, and the data dictionary. Together these describe the major behavior of the system and define a vocabulary describing the application and its environment. The textual requirements are hopefully part of the input to the requirements phase and it is often useful to elaborate these further mainly in the case where the original requirements are too unstructured or incomplete. The system operations are useful if the level of abstraction in the use cases are not found to be detailed enough.

In practise most of the time during the requirements analysis is usually spent designing the use cases, using these as a means to investigate the intended behavior of the system. It is impossible to completely cover the entire set of behaviors in the requirements analysis. Finding the right trade-off between the completeness of the use cases and the time and effort spent in requirements analysis is an important aspect of this phase. A recommended practise from [32] is that the use cases should cover 'approximately 80% of the primary scenarios along with a representative selection of the secondary ones'. The use case model (as well as the other models) will be further extended and refined during the elaboration phase so it is important to force the requirements analysis phase to a completion when a sufficient level of understanding has been reached.

System Analysis Phase

The system analysis phase takes over where the requirements analysis stopped. Where the requirements analysis focussed on *external requirements* the system analysis focuses on the *internal architecture* of the system. The analysis object model is the major medium used to describe the architecture. The architecture is defined in terms of the objects and object structures defined in this model. To establish the responsibilities of the different classes and verify that the proposed architecture is sufficient the use cases from the requirements analysis are in this phase refined in the analysis use case model.

It is not uncommon in practise that there is an overlap in time between the different phases and this is also true for the requirements analysis phase and the system analyses phase. It makes sense to start sketching the architecture of the system before the requirements analysis is brought to completion to capture ideas about the architecture whenever they emerge. It is however important to view this as a side activity and not forget to force the requirements analysis to a completion before digging into the details of the architecture.

The system analysis is finished when there is a consistent and well understood definition of the major objects needed in the system, including a clear definition of the responsibilities for the different objects, and the architecture is validated using the analysis use cases. It is beneficial if the system analysis can be performed by a small team consisting of the best people available including an experienced architect and some more developers.

System Design Phase

For many projects there is a need to divide the design work on more than one design team. In this situation the system design phase is a crucial phase since this is when the interfaces between the different parts of the system are defined. A good interface definition is essential to give the different development teams a possibility to progress with their work without confusion. In the system design the logical architecture from the system analysis is used to decompose the system into subsystems to be implemented by different teams and the interfaces between the subsystems are defined. The static interface is defined in terms of signals and/or remote procedure calls. The dynamic aspects of the interface are

defined in the design use cases that should be precise enough to be used as test cases by the design teams.

For practical reasons it is best if the same team that performed the system analysis phase also is involved in the system design phase, but extended with more designers and test engineers from the teams that will continue the design work to ensure that an understanding of the architecture is spread also into the different design teams.

The system design also includes planning the design module structure to be used in the design of the system to facilitate a smooth integration between the different subsystems. The design module structure is in essence a definition of the structure of the work to be performed by the different teams and it must be designed in order to facilitate reuse of common components and existing frameworks.

A third aspect to be considered in the system design phase, which is more of a project management nature, is to plan the following design and elaboration phases. The planning should be based on the achieved knowledge of the architecture and behavioral requirements and is intended to define the order in which the different features of the application is designed. The major criteria for defining this order is to minimize the risks involved in the project by validating the assumptions about the architecture as soon as possible and by designing the difficult and uncertain parts as early as possible. This implies that the design phase following the system analysis should concentrate on designing a first version of the application that covers an 'interesting' subset of the behavioral requirements (as defined in the use cases) and create a skeleton structure where all major components are in place but where their complete functionality is not yet realized.

When the system design is completed the tangible results are thus:

- A detailed architecture described using SDL block structures, including:
  - Static interface definitions with signals, remote procedures and the necessary data definitions, possible described using ASN.1
  - Design use cases describing the dynamic aspects of the interfaces
- A design module structure describing the structure that is to be used in the design phase

- A plan for the design process, that identifies key concepts to be designed and key use cases to use as a focus of the design phase and the elaboration phase

Design/Implementation Phase

The design/implementation phase, which as discussed above in general, is performed in parallel by several different development teams is in an SDL based SOMT project aimed at producing an executable SDL model of the application. Executable may here mean that it is possible to verify the design use cases against the SDL-model by executing the SDL model in e.g. a simulator or state space exploration based tool, but it may also imply an implementation activity that creates a first version of the application running in the target environment.

The choice of when to start the implementation activity is mainly risk-driven. If performance or memory requirements are crucial and seen as a risk in the project, e.g. if the hardware architecture is new and untested, then at least parts of the system should be implemented on the target to minimize these risks. If this is not the case, the integration with the target hardware can be planned to be part of some of the iterations in the elaboration phase.

Depending on the circumstances for the specific project the target platform can of course be very different, from the simplest case being that the target platform is the same type of computer as the development platform to the more complex situation where an embedded system is to be built and the target platform is a small micro processor. In the latter case, the implementation activity for an SDL-based project does in general consist of using a code generator to generate the code from the SDL system, write adaptation code to interface to the target hardware and the run-time environment to be used on this platform, and to use a cross compiler to produce code that is executable on the target machine.

In any case the deliverable from the design/implementation phase is an executable system that implements some of the requirements posed on the application and that is internally released as the first version of the application. The system should also have been through a testing process that verifies that the selected design use cases that where implemented indeed works as they should. This testing will of course consist of both a module testing performed on the different parts by themselves as well

as a system/integration test that verifies that the system as a whole works as it should.

The Elaboration Phase

The rest of the project, the elaboration phase, consists of a number of iterations through all the activities, from requirements analysis to implementation. Each iteration produces a new internal release of the application and refines and extends the architecture and system until it finally is finished. A typical iteration starts with a requirements analysis activity that includes a review and refinement of the use cases to be implemented in the current iteration. This is followed by system analysis and design activities that checks that the use cases are correctly distributed over the architecture and that the interface definitions are updated if needed. These activities should preferably be performed by an architecture group containing representatives both from the original architecture group who worked in the system analysis/design phase and from the different design teams.

The design and implementation activities that are part of an iteration are of course as before performed by the different design teams, each one refining and extending their part of the system to implement the use cases that are the goal of this particular iteration. The result of the design and implementation activity should be a new internal release of the application that has been through the appropriate module and system tests to verify its functionality.

In addition to extending the functionality of the system each iteration should also be focussed on minimizing the risks in the project. Crucial and difficult aspects should be tackled as early as possible in the elaboration phase to avoid unpleasant surprises at the end of the project.

Summary

A SOMT project can be described in terms of a number of phases, that describe the calendar time of the project. The phases are introduced as a means to get a visibility of important milestones and deliverables from the project. The various phases of a SOMT project are:

- Prestudy/conceptualization phase
- First iteration
  - Requirements analysis phase
  - System analysis phase
  - System design phase
  - Design/implementation phase
- Elaboration phase
  - Planned consecutive iterations

A graphical view of a possible SOMT project is illustrated in Figure 710.

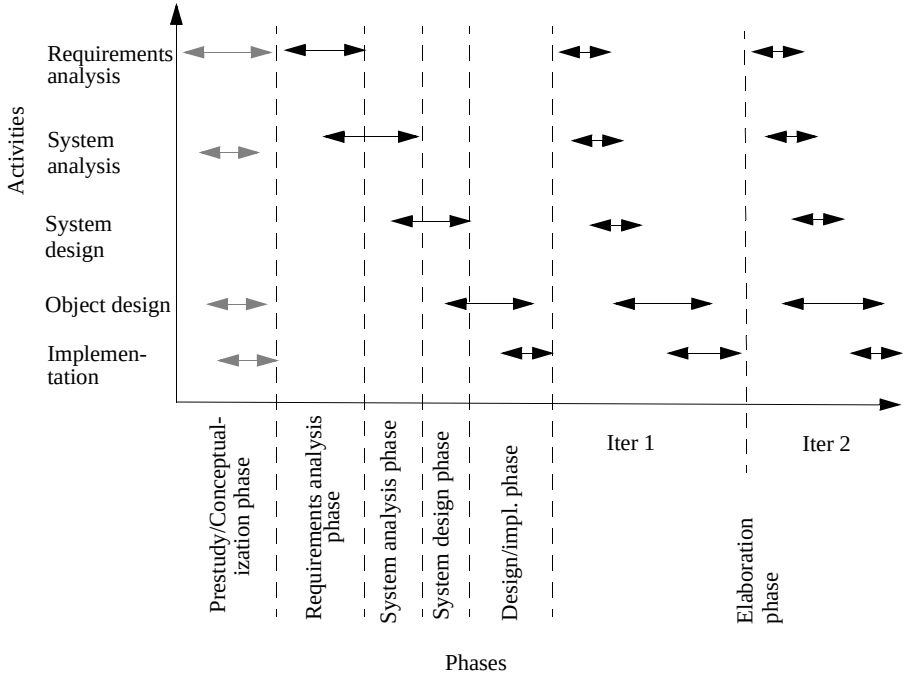

Figure 710: The different phases in a SOMT project

In this figure the various activities (as show on the vertical axis) are divided into a number of phases (as shown in the horizontal axis). The dashed lines indicate the deadlines for the different phases. Note that the activities associated with a specific phase may often start before the deadline of the previous phase is reached. This is common and useful. The purpose of the organization of the activities into phases is not to control when an activity starts, but to be able to focus on when an activity should be finished. This is what indicates the progress of the project.