Testing and Test Generation: State of the Art and Future Expectations

Anders Ek, Telelogic AB, Box 4148, S-203 12 Malmö, Sweden,
tel. +46 40 17 47 12, fax +46 40 17 47 47,
email anders.ek@telelogic.com

Abstract

Formal methods, testing and test generation are in this paper discussed from a pragmatical industrial perspective and in particular as seen from a CASE tool vendors point of view. Since a CASE tool vendor survives by convincing potential customers that they will make money by buying (sometimes expensive!) tools, he needs good sales arguments. So, how do formal methods, testing and test generation fit into this? Essentially the idea is to show that a development process supported by tools based on these concepts is more efficient, giving higher quality to a lower cost, than the currently used process .

The SOMT method provides such a process based on object oriented analysis and formal methods, and the requirements and testing track of this method is the main subject of this paper. As a complement to the method also the necessary tool support is discussed and exemplified with features from the Telelogic Tau tool set.

Keywords

MSC verification, test generation, use case, MSC, SDL, TTCN, UML

1 INTRODUCTION

Testing is an important part of any development project and this is true also for projects using object oriented techniques and an incremental, use case centred development method. In this paper I will discuss a development method with these characteristics. The focus of the discussion will be on the requirements and

A. Petrenko et al. (eds.), *Testing of Communicating Systems*

testing track that is a part of the method. The method is an elaboration of a tool specific method called the SOMT method (Telelogic, 1996) that is aimed at giving an efficient development process for certain classes of applications. The SOMT method is mainly intended for the development of reactive, distributed, real time, embedded and communicating systems. Characteristic for this type of applications is that they are difficult and expensive to test and that they very often are appropriate to design using object oriented methods.

The SOMT method is based on combining the strength of object oriented analysis and use case centred design with formal methods, specification level testing and test case generation. The notations used in this method are UML, SDL, MSC and TTCN but if needed it is easy to generalise the method to other notations provided they have the necessary level of formalism.

In this paper I will discuss both the method itself and the tool support required to use the method. When relevant I will give examples from the Telelogic Tau tool set (Telelogic 1998a). In section 2 I will give an overview of the method, in section 3 discuss various aspects of the requirements and testing track and finally in section 4 comment on the current usage in industry and the expectations we may have on the future development in this area.

2. THE SOMT METHOD

When discussing a development method it is convenient to discuss it in terms of activities and models, where the activities are different tasks that have to be performed and the models are the products (usually documents, source code etc.) that are produced by the activities. Even if the activities are presented here in one specific order this is just for the convenience of description. An actual project, in particular if it is using an incremental and iterative approach, will perform several activities in parallel and also iterate and alternate between different activities. Anyway, the following activities will have to be considered in a development project according to the SOMT method:

- requirement analysis,
- system analysis,
- system design,
- object design, and
- targeting/implementation.

Each of the activities has two different aspects, one architecture aspect that defines the structure and behaviour of the application and one requirements and testing aspect that defines how initial requirements are transformed into tests that are used to verify the correct functionality of the system. An overview of the

SOMT method is illustrated in Figure 1. In this figure the notation has also been assigned to the different. This is however only to give an indication of the most common usage, sometimes other variants are more convenient like using SDL in the system analysis or UML in the object design and the method does not exclude these possibilities.

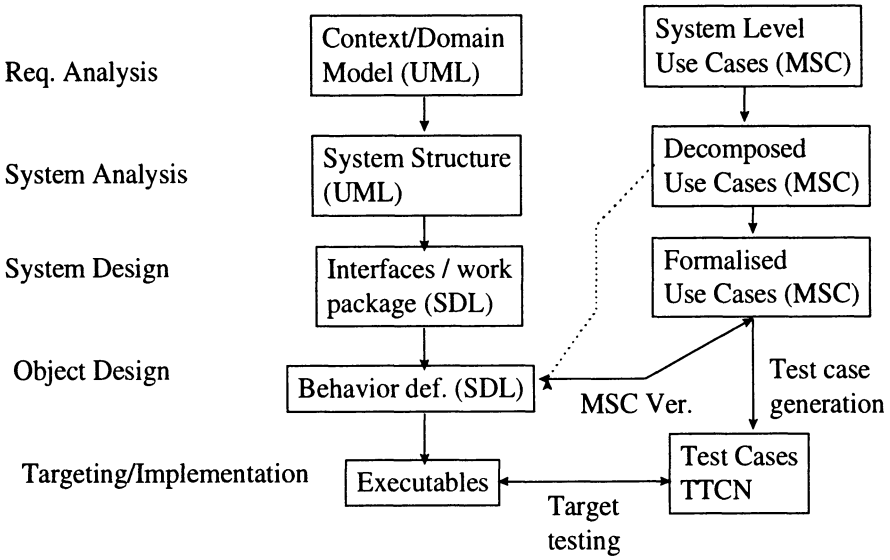

Figure 1. Summary of the SOMT method.

The rest of this section briefly describes the different activities.

2.1 Requirements Analysis

The requirements analysis is the first activity in the SOMT method. It is focused on the external aspect of the application to be built. The purpose of the activity is to capture and analyse the problem domain and the user requirements. For this purpose the system is viewed as a black-box and only the objects and concepts visible on the system boundary and outside the system are modelled.

For our purposes there are two essential models produced:

- a requirements object model including context diagrams and a problem domain model, and

- a use case model.

The requirements object model may be a conventional object model, i.e. one or more diagrams illustrating a number of objects and their relations (including inheritance and aggregation relations) but it may also be a fully described model, including behaviour, of the external view of the system. The purpose of the object model is two-fold:

- to use context diagrams to describe the system and the external actors that interact with the system, and
- to document all the concepts found during the requirements analysis and the relations between them in order to ensure that developers and users have a common understanding of the problem domain.

The use case model consists of a set of use cases, each described using MSCs and sometimes structured text. The purpose of this model is to capture and validate requirements from a user's point of view to make sure that the system will solve the right problem. As we will see below the use case model is also the first part of the requirements/testing track of SOMT.

2.2 System Analysis

In the system analysis activity the focus is on analysing the internal structure of the application. The purpose of the activity is to describe the architecture of the system and identify the objects that are needed to implement the required functionality. The models produced in this activity are:

- an analysis object model, to describe the architecture of the system in terms of objects and subsystems
- an analysis use case model which shows the interaction between the objects and subsystems in each use case.

The analysis object model is a conventional object model that forms the input to the object design phase. The analysts should in this activity be concerned with identifying the objects that must be present in the system, the responsibilities of the different objects and how they interact to fulfil the requirements posed on the system.

The analysis use case model is an elaboration of the use cases from the requirements analysis. In the system analysis the use cases are expressed in terms of the internal structure of the system and defines how the subsystems and objects in the system interact to provide the desired functionality.

2.3 System Design

The focus of the system design is on interfaces, reuse and work packages. The goal is to get an implementation structure that makes it possible for different teams to work on different parts of the system in parallel and to have precise definitions of their interfaces. To accomplish this the architecture track of the system design should product models like

- a design module structure, describing the source modules of the design,
- an architecture definition, containing SDL system/block diagrams, that defines the structure of the resulting application and that gives precise definitions of the static interfaces.

The requirements/testing track that is of more interest for us in this paper is in this activity focused on formalising the use cases to make them precise enough to be possible to verify against the application in a simulated environment. In other words, to prepare for the actual testing task.

2.4 Object Design

The purpose of the object design is to create a complete and formally verified definition of the system. The main work here is of two kinds:

- to do the coding of the behaviour of the objects in the system, and
- to verify that the system fulfils its requirements.

The models used in this activity are mainly SDL diagrams, in particular SDL process graphs that are used to define the behaviour of active objects.

The testing/verification task is done in a simulated environment using SDL simulators for interactive debugging and MSC verification for checking that the system fulfils its requirements as expressed in the formalised use cases.

2.5 Targeting/Implementation

The implementation and testing activities are aimed at producing the final application, i.e. executable software and hardware. The activities in this phase are very much depending on the execution environment of the application but usually include:

- either using an automatic code generation tool to produce the code from the SDL design or manually implementing the SDL design,

- integrating the code to the hardware requirements by means of using real-time operating systems and cross-compilers to generate the executable for the hardware,
- implementing and executing test cases in the target environment based on the design use cases from the system design activity.

From our perspective the most interesting part is the implementation and execution of the test cases. In the SOMT method this is done by generating TTCN test cases from the MSC use cases and executing the test cases against the application to verify that the targeting/implementation was successful.

3 THE TESTING TRACK IN SOMT

3.1 Use Cases

The testing track of the SOMT method starts with a use case oriented requirements capture. The term 'use case' has its origins in object oriented analysis methods and was established in the classical book by Jacobson (1992) but is actually an old idea that in different shapes has been practised for many years. The idea is to focus on the users of the system, called 'actors' in use case terminology. For each actor we analyse in what ways he would like to use the system. These different ways of using the system form the use cases.

So, each use case describes essentially a set of possible sequences of events that take place when one or more actors interact with the system in order to fulfil the purpose of the use case. A use case is thus simply a description, in one format or another, of a certain way to use the system. It has been found to be a very efficient way to capture a user's view of the system and the concept of use cases is now used in a number of object-oriented methods. A difference compared to most other approaches is that SOMT puts some more effort in the formalisation of the use cases to be able to use them for verification purposes during the object design. The formalisation is done using the MSC notation, in particular using the extensions provided in the 1996 version of the MSC standard as will be described in the next section.

3.2 Message Sequence Charts

A message sequence chart (MSC) is a high-level description of the message interaction between system components and their environment. A major advantage of the MSC language is its clear and unambiguous graphical layout which immediately gives an intuitive understanding of the described system behaviour. The syntax and semantics of MSCs are standardised by the ITU-T as recommendation Z.120.

There are various application areas for MSCs but for our purposes in the SOMT method the most important are:

- to define the requirements of a system,
- to check the consistency of SDL specifications using MSC verification, and
- to form as a basis for specification of TTCN test cases .

There have been two versions of the Z.120 recommendation published, one in 1992 and one in 1996. The 1992 version included the conventional sequence diagram notation that made it possible to essentially describe simple sequences of events. This recommendation was quickly implemented in a number of commercial tools but two problems were reported from people using MSCs in industrial applications. Both problems concerned how to handle complexity in the requirements specifications.

The first issue was that when people started using MSCs they quickly found out that they wanted to decompose the MSCs into smaller MSC that could be reused in different ways to cover different cases. In MSC'92 it was suggested to use the condition symbols for this. If one MSC ended with a condition 'Idle' and another started with the same condition it was informally meant that the second could follow the first one. The problem was that it was difficult to get a high level view of the actual use cases and the overview was lost.

The second problem was that the number of MSC that was needed to describe the requirements was found to be very large. To a large extent the reason for this was that all different combinations of exceptional cases and alternative possibilities had to be described in separate MSCs. This lead to either an explosion in the number of MSCs needed or the introduction of informal comments into the MSCs. Most actual users chose the informal comment solution. Unfortunately this made all verification tools useless.

However, both of these problems were solved in the revised Z.120 recommendation issued in 1996. This was mainly accomplished by introducing two new concepts:

- high-level MSC diagrams and
- inline expressions.

The high-level MSC diagrams (HMSCs) solves the overview problem. The HMSCs simply describe how other MSCs (either MSC'92 style MSCs or HMSCs) can be combined to define a more complex behaviour. For example, the HMSC in Figure 2 shows a situation where we first have the MSC 'ConnectionEst' and

then one or more 'DataTrans' followed by one 'Disconnect' MSC. Note that when the flow lines between the MSC references branch, this indicates an alternative between the subsequent MSCs.

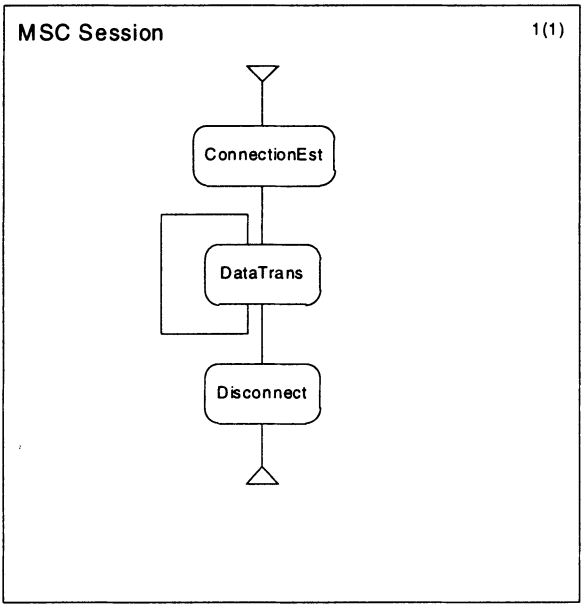

Figure 2. A high-level message sequence chart.

The combinatorial explosion of the number of MSC caused by similar, but not identical MSCs was solved by the addition of inline expressions in MSC'96. The idea is that a number of the messages and other events in an MSC can be framed and defined to be e.g. optional or an alternative to other events. Essentially this gives us a possibility to describe several similar, but slightly different, MSCs in one MSC. The inline expression possibility reduces the number of MSCs that need to be written substantially while they still preserve the intuitive syntax of MSC'92 and the formality of the requirement specification

From the SOMT point of view the new 1996 version of MSC meant that the developers in industry could start capturing the full set of requirements and still remain in the formal MSC notation.

3.3 Decomposing the MSCs

In the requirements analysis the MSCs are usually on an application level of abstraction with one instance axis for each actor and one instance axis for the system to show the external view of the use cases. In the system analysis the MSCs are elaborated and the system instance axis is replaced with instance axis for the different subsystem and/or objects in the system. the purpose is to see how the functionality of the use case is distributed among the subsystems.

3.4 Keeping the MSCs Consistent and Formal

In the system design activity the designers of the application formalises the interfaces between different subsystems. Some typical tasks done is to precisely specify signal lists, signal names, parameters of signals and the structure of data.

In the testing track the main tasks are to keep the use case MSCs consistent with the precisely defined static interfaces and to do a thorough review of the use cases to make sure that they are precise enough both to act as an correct requirement for the implementers of the system and as a basis for testing of the requirements.

This may sound like a trivial activity but it involves quite a lot of work and it is very important if we want to get a smooth development and testing in the rest of the project.

3.5 Design Level Testing and MSC Verification

In the object design activity there are two major tasks to be performed. First of all the details of the application is defined in SDL. This means in practise that the complete dynamic behaviour of the objects in the system ('processes' with SDL terminology) is coded using the state machines, abstract data types and the other concepts of SDL. The result is a complete definition of the application described as a set of communicating SDL processes whose combined behaviour is defined by the SDL run time model. Since SDL includes a precise definition of the run time semantics it is fairly straight forward to build tools that can simulate an SDL system using the predefined run time semantics. The Tau tool set contains both SDT Simulator for debugging SDL systems and tools like the SDT Validator that can automatically explore the state space of the SDL system.

This executable property of SDL is used in the SOMT method for the requirements/testing task to be performed in object design; the verification of the use case MSCs against the SDL specification. Basically this is a kind of testing performed in a simulated environment with the purpose of verifying that the different execution paths described by the MSCs indeed are implemented in the application. This is most efficiently done by simply letting a state space exploration tool perform a search through the state space and in parallel evaluate

the MSC seen as a restriction on the possible execution sequences. In the Tau tool set the SDT Validator is designed to do this (See Ek, 1993).

This design/specification level debugging is very convenient in particular for distributed systems where a test in a real target environment would include downloading the application in a distributed testing hardware which is a fairly complex process. Also for real time systems the possibility to test in a simulated environment is very efficient since real time application often involve specific hardware that may not be available until late in the project. By testing in a simulated environment the testing process can start earlier.

3.6 Test Case Generation

However, even if the design is verified in a simulated environment there still is a need to do a real testing in the target environment. To facilitate this the SOMT method uses the TTCN language as test notation for target testing.

The generation of TTCN test cases from the MSC use cases is a problem that has some intricacies that may not be immediately noticed. One is that an MSC use case may, in particular if HMSCs are used, actually be a description of many test cases. The reason is that the MSC may branch and the decision of which branch to choose depends on what aspect of the use case we want to test. This is a very convenient feature in the MSC since it gives a compact definition of the use case and a state space exploration based MSC verifier can deal with this. However it is a problem when trying to test in a target environment and thus it is a complication when mapping to TTCN.

Another problem is that often the MSC use cases doesn't contain a complete description of e.g. all data values to use when sending signals to the application. The MSC verification can handle this but the data values are needed when doing real testing on target.

To overcome these problems a flattening and further formalisation of the MSC use cases are needed before moving to TTCN. In the Tau environment this can be accomplished by using the MSC verification to produce test purpose MSCs as a side effect of verifying the use case MSC. The test purpose MSCs are simple MSCs that contain no branching but sufficient details to act as input to the TTCN generation. This is illustrated in Figure 4.

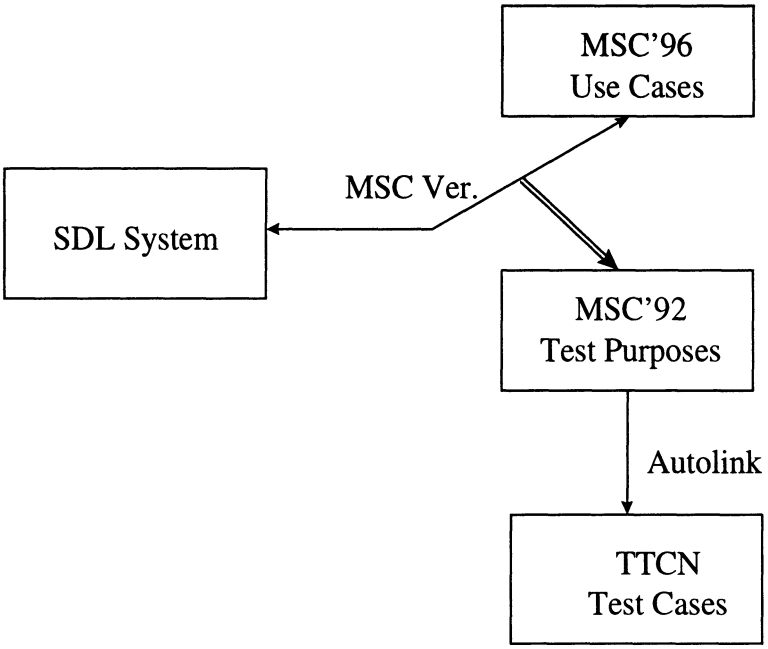

Figure 4. Generation of test purpose MSCs and TTCN test cases based on use case MSCs.

Another complexity arises from the circumstances under which the generated TTCN test cases should be used. If they are only to be used for an in-house automatic verification step, where the generated tests are executed on the run time platform in order to check that the adaptation of generated code works as expected, then the quality in terms of readability of the test suite etc. is not too important. However, if parts of the SDL system are implemented by hand or maybe even by a different organisation than the one that develops the SDL and TTCN the situation is different. Then the TTCN specification will be official and thoroughly inspected by people. In this situation it is essential that the TTCN test suite is nicely structured and readable. To a large extent the readability of a test suite depends on minor details like the names of constraints and parameters. Unfortunately this structuring and naming is very difficult or impossible to generate automatically. Fortunately, quite a lot of effort has been put into solving this by allowing the user to guide and control the automatic generation of test cases including e.g. controlling the naming and parametrisation of constraints.

The Autolink feature in the SDT Validator (see Ek *et al* 1997, Schmitt *et al* 1997 and Telelogic 1998b) is the result of a joint project between Telelogic and the Institute for Telematics in Luebeck. This project has to a large extent been

focused on these practical aspects of the test generation problem. The basis of the Autolink test generation features is the state space exploration possibilities in the SDT Validator but a substantial amount of the actual work has been to develop methods that solves the more practical problems. The result is a product that now is used to produce TTCN test cases to be published as official ETSI recommendations (see Schmitt *et al* 1998).

4 STATUS AND EXPECTATIONS IN INDUSTRY

The SOMT method has been the recommended methodology guideline for people using the Tau tools since it first was published in Telelogic (1996) and the combined usage of UML, SDL, MSC and TTCN is now spreading in industry.

SDL is now an accepted development notation in the telecom part of industry where many applications have been developed using automatic code generation from SDL designs. It is also used in standardisation where SDL is used as a specification notation, e.g. the INAP specification done by ETSI is specified in SDL. Nowadays, the tool support for SDL based development and code generation from SDL are very good and both our Telelogic Tau tools and other tools like Verilogs ObjectGeode are now successfully competing on the CASE tools market.

The combination of UML for analysis and SDL for design is starting to get used but is not really wide spread yet, even though predecessors of UML and object oriented analysis has been used for many years as a complement to SDL in the early stages of development. However, since UML now is standardised and is replacing older techniques like OMT and Booch I would expect the usage of UML to increase very rapidly in the future.

MSCs and similar notations have been used for many years in industry to capture requirements and is an established method. Traditionally however MSC has been used in a fairly informal way making it difficult to use it directly in tools for verifying consistency with the design. To a large extent this is the case because the first formal version of MSC as standardised in 1992 did suffer from a number of weaknesses compared to the informal versions used in industry. As a consequence MSC is heavily used but today mainly to informally specify the requirements and also often extended with informal comments to overcome the limitations in MSC'92. The typical SDL developer of today would use the MSCs as input to a manual simulation task. He would take the MSC and manually run the SDL simulator according to the MSC requirements, creating simulation scripts that can be reused for regression testing. This is not a bad strategy but involves a substantial amount of manual work and, which may be a more severe problem, it makes it necessary to maintain two levels of manually created and

maintained descriptions. Both the requirement MSCs and the simulation scripts need to be maintained. As the new 1996 version of MSC is starting to get tool support and acceptance in industry this situation has started to change and MSC verification is beginning to be used as an industrial testing technique (see e.g. Ek *et al,* 1997).

On the target testing market most major telecom suppliers today have had their own proprietary test notation and test environment. However, as more and more commercially available test platforms based on TTCN are released and new test specifications using TTCN are published we have at Telelogic seen a clear trend towards TTCN as a platform for testing tools. This has raised the interest for automatic generation of TTCN code considerably since there is a very strong request from our customers to try to reduce the manual work needed for developing tests.

To summarise the industrial perspective it is, as always, driven by commercial considerations. To make money from new products they have to be delivered on time, at reasonable price and quality. All of these aspects lead to a need to streamline the development process and to use tool support to automate as much as possible of the necessary task in the development process. It pays off to let the engineers do engineering and use computers for routine tasks.

As a consequence all new CASE tools features that promise to cut down the manual development costs are very interesting for the development industry. This is good news for example for test generation tool developers, since the test generation tools definitely match this description. However, there are two challenges facing the introduction of the new tools:

- They must be able to handle industrial size problems.
- They must fit into the development processes used in industry.

Both of these problems are difficult, but with new commercial tools like the Telelogic Tau tools that are able to handle real applications, but still are based on formal techniques, the first problem is beginning to get a solution, even if there of course still is some potential for improvement.

The second problem, i.e. to change the way of working for a development department with maybe several hundred developers, a large design base of existing code and several already released product versions is often an even more challenging problem. Fortunately, from a CASE tool vendors point of view, the development departments are also faced with tough requirements from their respective customers. They need to shorten their development time and increase their productivity with a maintained high quality. This is usually not possible

without changing the work process and in fact the entire CMM and quality certification movement is targeted on making the development processes visible so they can be enhanced and made more efficient. This gives CASE tools that try to automate e.g. the testing, a very good opportunity to be introduced as part of a more streamlined development process.

So, the need and the market opportunity for design level testing, test generation and efficient test environments are here and it is up the tool vendors and testing community to face the challenges and provide the solutions.

6 REFERENCES

Ek, A. (1993), Verifying Message Sequence Charts with the SDT Validator. In *Proc. 6th SDL Forum*, Darmstadt, 1993, North-Holland.

Ek, A., Grabowski, J., Hogrefe, D., Jerome, R., Koch, B. and Schmitt, M. (1997) Towards the industrial use of validation techniques and automatic test generation methods for SDL specifications, In *Proc. 8th SDL Forum*, Evry, France, Elsevier.

Jacobson, I. et al (1992), Object-Oriented Software Engineering, Addison-Wesley.

OMG 1997, *The Unified Modelling Language Version 1.1*, Object Management Group.

Schmitt, M., Koch, B., Grabowski, J., Hogrefe, D. (1997) Autolink - a tool for automatic and semi-automatic test generation. *In Proceeding of the Seventh GI/ITG Technical Meeting on Formal Description Techniques for Distributed Systems*, Berlin, June 1997.

Schmitt, M., Koch, B., Grabowski, J., Ek, A., Hogrefe, D. (1998) Autolink - Putting test generation into practise, In *Proc IWTCS 98*, Tomsk, Russia, Chapman & Hall

Telelogic (1996), Tau methodology guidelines part 1 - The SOMT method, *Telelogic Tau 3.1 Documentation*, Telelogic.

Telelogic (1998a), *Telelogic Tau 3.3 Documentation,* Telelogic.

Telelogic (1998b), Validating an SDL System, *Telelogic Tau 3.3 Documentation,* Telelogic.

7 BIOGRAPHY

Anders Ek has studied computer science and engineering at Lund Institute of Technology in Sweden. After his graduation in 1986 he started working at Telia, the national Swedish telecom operator. At Telia he did research on formal methods and associated development methodology. Since 1992 he has been working with development of tools and methods at Telelogic, a commercial CASE tool vendor providing tools based on formal description techniques.