A. Wesslén and C. Wohlin, "Modelling and Generation of Software Usage",
Proceedings 5th International Conference on Software Quality, pp. 147-159,
Austin, Texas, USA, 1995.

Modelling and Generation of Software Usage[1]

Anders Wesslén and Claes Wohlin
Department of Communication Systems, Lund Institute of Technology
Lund University, Box 118, S-221 00 LUND, Sweden
Phone: +46-46-222 3319, Fax: +46-46-145823, E-mail: (wesslen, claesw)@tts.lth.se

Abstract

Usage testing or operational profile testing seems to be the general trend in testing software systems, but it is not enough. This paper presents briefly a view which extends the usage testing concept into other software evaluation techniques than traditional testing. It is also shown how software usage can be generated semi-automatically for use throughout the software life cycle. The generation of software usage is made by modelling usage with a state hierarchy and then by transforming it into a usage model in a well-defined description language.

Keywords

Usage testing, usage modelling, test case generation, validation and verification, reliability certification

1. Introduction

One of the most important quality attributes of software systems is software reliability. The reliability of software systems is becoming a more and more critical issue as our dependency of these systems grows rapidly.

Software testing is an important activity in software development as it is the last resort to stop poor software from entering the field. Therefore, much attention is paid to test techniques that try to resemble the operational phase in order to find the faults that are most disturbing to users of the system, i.e. failures that influence users perception of system reliability the most. This type of testing is often referred to as statistical usage testing [Mills88, Runeson93] or operational profile testing [Musa93].

This paper presents briefly some ideas of how the testing approach can be extended in a life cycle perspective, hence supporting a usage evaluation approach through, for example, dynamic analysis or simulation of the software design. The extension means that the validation and verification is a process that follows the software development throughout the system's life cycle. The constituents of such a process are discussed briefly and then the focus is set on developing a model of the expected usage of the software system and how software usage can be generated. In particular, the possibility for semi-automatic generation of software usage based on a hierarchical model of the usage is discussed in this paper.

The usage validation and verification process are discussed briefly in Section 2. The development of a model of the expected usage (usage modelling) is described in Section 3 and the

1.This work is supported by National Board for Industrial and Technical Development (NUTEK), Sweden, reference Dnr: 93-2850.

semi-automatic generation of software usage is discussed in Section 4. Finally some conclusions are presented in Section 5.

2. Validation and verification

2.1 Usage testing and extensions

Usage testing [Mills88, Runeson93] and operational profile testing [Musa93] are discussed as methods to improve software testing. The objectives of both these approaches are to detect and remove faults which influence the reliability of the software the most and also to allow for reliability certification during the testing process. A prerequisite to achieve these objectives is that test cases must be generated according to the anticipated usage in operation. A high level abstraction of the usage testing process is illustrated in Figure 1.

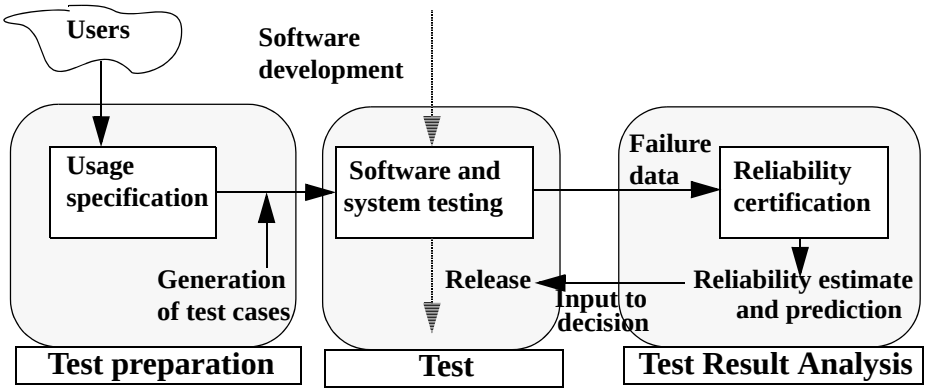

FIGURE 1. High level abstraction of the usage testing process.

The usage testing process is divided into three major parts: test preparation, test and test result analysis. The expected behaviour of the users is described in a usage specification from which test cases must be generated. The software development is done in parallel with the derivation of the usage specification, which consists of a usage model and a usage profile. The usage testing process intercepts with the development process at the testing phase. Failure data are collected from the test phase and the data are put into a suitable software reliability model; see for example [Musa87], to certify if the required reliability has been reached. The certification includes both estimation and prediction of software reliability. The result from the certification procedure is one input to make a decision whether to release the software system or not.

The usage testing concept can be extended to cover more activities than traditional testing. A method for usage analysis of design descriptions was proposed in [Wohlin92] and a related idea for simulation is discussed in [Wohlin94a]. The latter is primarily intended for performance simulation, but the method can be enlarged to take reliability issues into account as well. Therefore, it is more suitable to discuss a Usage Validation and Verification (V&V) Process, where the middle box (testing box) in Figure 1 can be changed to software and system V&V.

The above means that the usage V&V process intercepts with the software development process at several phases, for example analysis of software design [Wohlin92], simulation of software design (either purely functional simulation of the design or a combined functional and performance simulation where the software design executes on a simulation model of the target system) [Wohlin94a] or usage testing, as discussed above. These other possibilities of applying a usage approach are discussed elsewhere, hence the objective here is to illustrate

how software usage can be generated to support the usage approach independently of the phase of the software life cycle.

2.2 Usage V&V process

2.2.1 V&V preparation

During this phase of the V&V process, there are two tasks that must be done, first to derive a usage specification and second to generate the anticipated usage of the software. Deriving a usage specification is also called usage modelling.

Usage specification

The expected users and their anticipated usage of the software must be investigated. The analysis is the basis for formulating a usage model, which must be described with a well-defined description technique. Some different techniques have been proposed as for example Markov chains [Whittaker93], a state hierarchy model [Runeson92] and a tree structure [Musa93]. The hierarchical model are used in this paper and the technique is briefly introduced in Section 3.2. The usage model is complemented with a usage profile, which describes the anticipated behaviour in terms of relative probabilities for use.

Generation of software usage

The basis for generation of software usage is the usage specification. The generation results in a list of user stimuli and the expected responses from the system, henceforth referred to as a V&V case. The expected responses are used to evaluate if the V&V case has succeeded or failed. By making transitions in the usage model according to the usage profile, the V&V cases are generated.

The idea in this paper is to transform the usage specification to a representation that can be executed. By executing the usage specification the roles have changed, i.e. the usage specification can be seen as a new system while the person performing the generation behaves as the expected behaviour of the implemented software. The person generating the V&V cases uses the requirements specification to evaluate the user stimuli and to respond with the right system responses. The V&V cases are hence generated semi-automatically by logging the execution of the usage specification (see Section 4.2). The term semi-automatically is used as the V&V cases are logged automatically, but a person acting as the system is needed to obtain system results. The person acting as the system is henceforth referred to as an oracle because he (or she) knows the correct responses for the inputs to the system. Automation of the oracle based on a formal requirement specification is an area for future research.

It may be the case that V&V cases can be generated semi-automatically from whatever representation is used in the usage specification. This does, however, require tool support and two options can be identified:

- tool support is available or developed for the representation used,
- tool support is not available for the used representation, but it is possible to translate the current specification easily into a representation for which tool support is available.

The latter approach is used here as it was more important to illustrate the potential of the method than to develop specific tools.

The method presented in [Wohlin92] is exemplified with SDL (Specification and Description Language) [Belina91] and SDL is standardized by the telecommunication standardization sec-

tor of the International Telecommunication Union (ITU-T), which provides standards for primarily the telecommunication domain, although the techniques standardized can be used for other applications as well. SDL has been used as a suitable design representation of the usage model, as tool support is available and the technique has the necessary concepts to describe software usage. Thus, the state hierarchy model is translated into an SDL description. The translation is easy as both state hierarchy model and SDL are based on finite state machines. If the software is designed with SDL, then the choice to use SDL for usage modelling is particular beneficial, [Wohlin92].

2.2.2 V&V execution

During this phase the software system is validated and verified with the V&V cases. The software system is executed according to the V&V cases and the system's responses are compared with the expected responses in the V&V cases. The evaluation can be made automatically if the V&V cases are described in the same language as the software system (see Section 4.3), or if the necessary support is supplied by other means. If the system's response differs from the expected a failure has occurred. The failure has to be analysed carefully. The fault may be either in the software implementation or the V&V case. The failure data are collected and are passed to the V&V result analysis phase.

2.2.3 V&V result analysis

The collected failure data from the V&V phase are analysed during this phase. The failure data can be used in reliability models to estimate and predict the reliability of the software system. The output from the reliability models is used as a metric for acceptance or rejection of the fulfilment of the reliability requirements. In the early life cycle phases the estimations also provide a means for planning and controlling the forthcoming software development.

2.2.4 Method summary

1. Model usage with a state hierarchy model.
2. Add a usage profile to the model.
3. Transform the state hierarchy model into a usage specification in SDL.
4. Execute the usage specification to generate V&V cases, representative of the software usage.
5. Verify and validate the software system with the V&V cases.
6. Collect failure data.
7. Estimate and predict the reliability of the software.
8. Accept, reject or continue the verification and validation based on the outcome of the execution, or if being in an early phase of the development then the information can be used to take informed management decisions of the future work.

Step 1-4 are further described in Section 3 and Section 4. Step 5 corresponds to the same activities as when performing verification and validation with other methods. Finally, step 6-8 are treated elsewhere; see for example [Wohlin94b] or for a general description of these type of techniques [Musa87].

3. Usage modelling

3.1 Introduction

Usage modelling is an essential part in usage testing [Mills88, Runeson93] as well as in operational profile testing [Musa93].

Usage modelling means deriving a usage specification, which can be viewed as consisting of two parts, namely a usage model and a usage profile. The model describes the usage from a structural point of view, while the profile contains the usage probabilities, hence providing a statistical view of the usage. These two parts of the usage specification are discussed subsequently.

3.2 Usage specification using a state hierarchy model

3.2.1 Introduction

This section gives a brief introduction to the state hierarchy model and how to make the usage specification from this concept. The model used is called the state hierarchy (SHY) model, and it is described in more detail in [Runeson92].

3.2.2 Usage model

The state hierarchy (SHY) model is introduced to cope with modelling complex systems with several user types and numerous different users. The objective of the model is to divide the usage modelling problem into different levels, hence focusing on one aspect at the time. The number of levels in the model can easily be adapted to the needs when modelling.

Here a five level hierarchy is assumed. The first level in the hierarchy is a common usage level, the next level is supposed to describe the different user types of the system, the third level shall model the actual users, while the fourth models the services provided to the users and finally the fifth level describes the specific behaviour of the service as seen from a user´s point of view. The behaviour level is described with an ordinary Markov chain. An optional sixth level may be added to cope with a specific part of the behaviour. Therefore, the sixth level is referred to as the sub-behaviour level. The latter means specializing a specific behaviour which tends to make the behaviour level grow too large. The SHY model, without the sub-behaviour level, is illustrated in Figure 2.

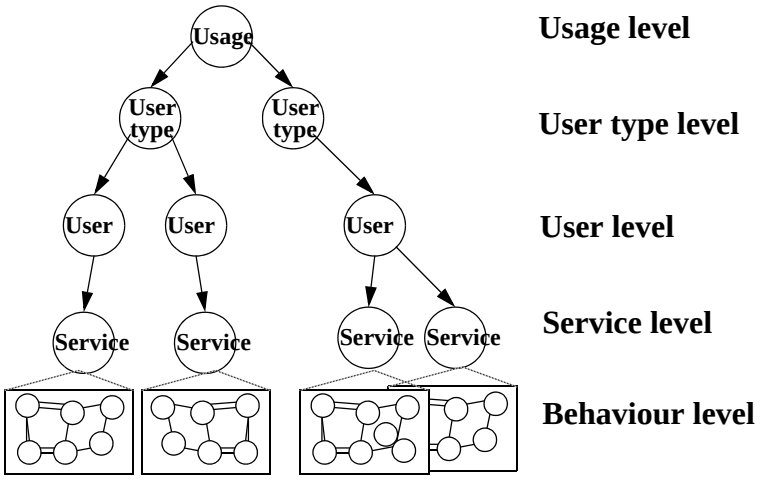

FIGURE 2. Illustration of the SHY model.

Dependencies between users and between different services available to a user are modelled with links, i.e. references between different parts of the SHY model.

The hierarchy means that a service used by several users is only modelled once and then instantiated for all users using that particular service. The generation of V&V cases according to the anticipated software usage are made by running through the state hierarchy. The next event to be added to the V&V case is generated by first choosing a particular user type, then a specific user of the chosen type and finally a service is chosen and based on the state of the chosen service a transition (event) is added to the V&V case.

3.2.3 Usage profile

The usage profile must describe the anticipated usage of the system as it is put into operation. The SHY model assumes a division of the usage profile into two parts, namely individual profile and hierarchical profile.

The individual profile describes the usage for a single user, i.e. how a user behaves when using the available services. All users of a specific user type have the same individual profile. This profile refers to the transition probabilities on the behaviour and sub-behaviour level and the probabilities are assumed to be static.

The hierarchical profile is one of the major advantages with the SHY model as it allows for dynamic probabilities. It is obvious that it is more probable that a subscriber, connected to a telecommunication system, who has recently lifted the receiver dials a digit, than that another user lifts the receiver. This means that the choice of a specific user to generate the next event depends on the actual state of the user. This is handle by introducing state weights which model the relative probability of generating the next event compared with the other states of the service. The state weights are the basis for the hierarchical profile and the probabilities in the hierarchy are derived from the weights. This is further discussed in [Runeson92, Wesslén93]. The dynamic probabilities are easily included in the V&V case generation procedure. This opportunity is not explicitly handled in any of the other models describing usage.

3.2.4 Formulation of the usage specification

Usage model

To be able to define the SHY model the services and the expected users must be analysed. The SHY model defines services and their behaviour as seen from an external user, as well as the type of users and their available services. The definition of the SHY model is made in the following steps:

- Service level

  First, the available services are identified. Often, at least in telecommunication systems, an external service of the system can be decomposed into three system services, the behaviour, order, and cancellation of the external service. These services define the service level in the SHY model. For each service in the service level, it must be decided if the user can observe the service’s behaviour. If the behaviour is not observable from the user, the service does not have to be modelled and the service can be removed from the level.

- User level

  The services a specific user can use must be defined. The users define the user level in the SHY model.

- User type level

  The users are grouped together based on which services they use and the individual profile of the users. The groups of users define the user type level in the SHY model.

- Usage level

  All user types are grouped together into one group representing all the usage of the system. This group is the state at the top of the hierarchy.

- Behaviour level

  In the SHY model the behaviour of the services is described in the behaviour level. The usage of the services is described in the behaviour level with a Markov chain. The Markov chain consists of states and arcs between states. The states in the Markov chain are those states that are observable for the user of the service. The arcs between the states define possible transitions between the states. Possible transitions are transitions that the user can make by himself or transitions that depend on other services or users.

Links

To describe dependency between different users, links are introduced in the SHY model. A link is a dependency that models that if one transition is made in one service, it triggers a transition in another service. The dependencies between users, and hence the links in the model, are best handled by using the responses from the system. The responses trigger transitions in the usage model.

For example, in a telecommunication system one user can call another user and when the user has dialed the telephone number it starts to ring in the other telephone. In this example, when the user has dialed the telephone number the user comes to a state waiting for a response from the system, e.g. a ring tone or a busy tone. If the other user is idle the system responds with changing the dialing user's state to ring tone and the dialed user's state to ringing.

A user can use more than one service and these services may depend on each other. This type of dependency is expressed as direct links between the services.

Stimuli

When the user interacts with the system, the user sends stimuli to the system and the system responds by changing its state. A stimulus is an event that the user generates, for example the user lifts the receiver in a telecommunication system. The stimulus from the user is assigned to a transition in the behaviour level of the service. If the stimulus for one transition must be refined, one can describe the choice of the stimulus with a Markov chain. This refinement is described in a sub-behaviour level. For example, when dialing a telephone number in a telecommunication system, the choice of a digit may be described with a sub-behaviour level.

Usage profile

The usage profile for the SHY model is, as stated above, divided in two parts: individual profile and hierarchical profile.

These profiles must be assigned values, i.e. probabilities for single events by an individual user and the state weights, which models the relative frequency of events, are needed. The state weights form the basis for determining the hierarchical profile.

An common objection concerning the profile is that it is impossible to determine the probabilities. It is, of course, true that it is not possible to determine the probabilities with high accuracy,

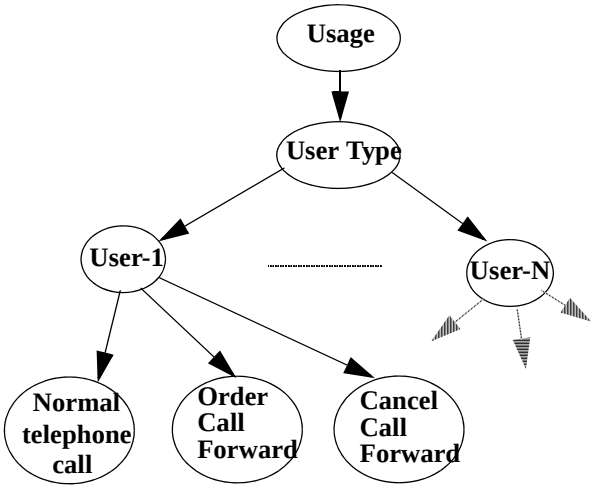

FIGURE 3. The SHY model for the example.

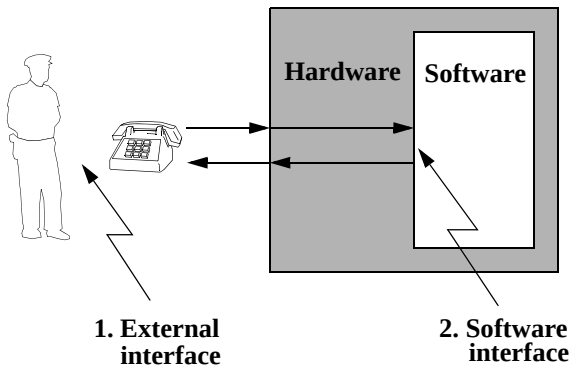

FIGURE 4. Illustration of V&V interfaces.

but it ought to be possible to divide the probabilities into a number of classes, hence obtaining a better picture than by just ignoring the actual usage.

Both the probabilities and the state weights must be based on market surveys, experience from earlier systems and expert opinions. The latter including personnel from both the developer and the procurer of the system.

3.2.5 A short example of the usage model

The example is a very simple telecommunication system, which has a few services implemented. The services are a service for normal telephone call and a service to do call forwarding. Every user in the system can use the services and they use them in the same way. The call forwarding service is divided into three different services, ordering, cancellation and the actual behaviour of the service call forward. All services are observable for the users except the behaviour of call forward. The resulting SHY model for the usage of the system is shown in Figure 3. The behaviour levels are left out in the figure.

4. Generation of software usage

4.1 Introduction

The V&V cases are directed towards the part of the system to be evaluated, i.e. it is the usage of the parts to be analysed that should be modelled, which may mean that all users are other parts of a larger system. For example, the evaluation of software components to certify their reliability is discussed in [Wohlin94b]. The part to be evaluated is referred to as the V&V object. V&V cases must be generated from an interface and some different interfaces may be considered. In particular two specific interfaces are easily identified, i.e. interface to the actual user and software interface. These interfaces are illustrated in Figure 4 with a telecommunication system.

The first interface is the external interface (denoted 1.), where V&V cases are generated from the actual user interface. The external interface means that the software is executed on the target machine. The second interface is the software interface (denoted 2.), which means that the software is executed (analysis or simulation) in the development environment.

It is advantageous to use the software interface since it is easier to generate the usage semi-automatically, as more information is available through the software signals than in the external interface. On the other hand, using the software interface is only a partial evaluation since failures may occur as the software is loaded on the target machine. The solution is to use both interfaces, i.e. the software must first be evaluated thoroughly in the development environment and then as it works, it can be evaluated on the target machine.

The most extensive evaluation must be performed in the development environment since evaluation in a real environment is much more expensive. It must be observed that V&V cases generated at the software interface may be used in the V&V process at the external interface, since the V&V cases may be generated with a description in natural language. Henceforth, the software interface is assumed unless explicitly stated otherwise.

There are two issues that must be stressed when generating software usage. Firstly, it is how to obtain the expected system responses from the user stimuli. The expected system responses are used to evaluate if the V&V have succeeded or failed. The second is how to generate the large number of V&V cases that is required to obtain the desired accuracy of the reliability measurements.

Using the software interface enables the usage specification to use the system responses to guide the usage of the system. This means that the usage specification stops for system responses in certain points in the usage, e.g. if you pick up the telephone receiver, you expect that the system responds with a dial tone before you do anything else. To get the next user stimuli, the previous user stimuli must be evaluated and the right system response must be noted.

The second issue is solved with semi-automatic generation of software usage. The generation is done by executing the usage specification with the input of the expected system responses. The V&V cases are the logged stimuli and expected system responses. The execution of the usage specification can be made in two different ways, as discussed above.

Verifying and validating the software in the development environment, and if the V&V cases are described in the same language as the system, enables that the validation and verification can be automatically (see Section 4.3).

4.2 Semi-automatic generation of software usage

The idea of semi-automatic generation is to execute the usage model according to the usage profile and logging the user stimuli and the system responses. To be able to execute the usage model, it must be transformed into a representation that can be executed, if tool support is not directly available, which it is not in this particular case. It is preferable to use a representation that supports the basic ideas of the usage specification and has a well working tool environment. In the work in [Wesslén93] the usage specification is based on the ideas of finite state machines and SDL, [Belina91], was therefore chosen as the executable representation. The tool environment used for SDL supports verification of the model, both syntactic and semantic. It also has tools for generation of C-code, simulation and dynamic analysis of the model. The dynamic analysis tool is of special interested, if the usage specification is to be used in usage analysis of the system's design [Wohlin92]. These existing tools open up the opportunity to extend the V&V process beyond testing. Thus, the usage approach can be applied earlier in the software life cycle as stressed as important in Section 2.

The usage specification made with the SHY model is very simple to transform into an SDL model. The state hierarchy of the SHY model can be viewed as a tree, with the top level, usage

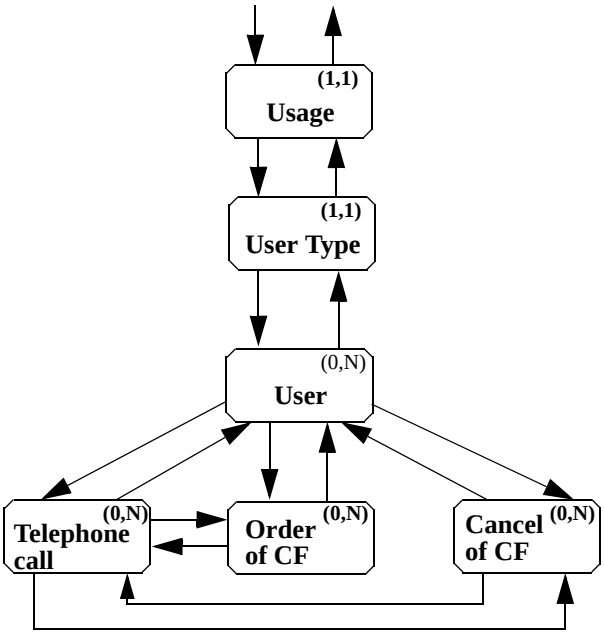

FIGURE 5. The usage model in SDL.

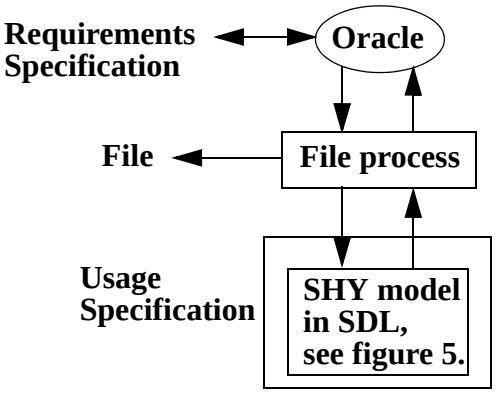

FIGURE 6. Generation of software usage.

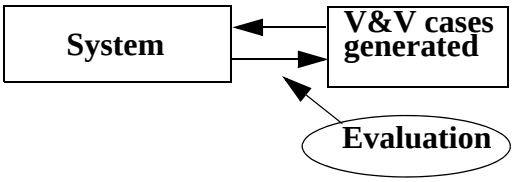

FIGURE 7. V&V cases execution and evaluation of the results.

level, as the root. Every node in the tree is represented in the SDL model with a process. The leaves in the tree are the services and each service forms a process. The service process is designed according to the behaviour level in the SHY model. A service's behaviour level is a Markov chain, described with states and transitions between states. A process in SDL is a state machine, which is described with states and transitions from one state to another. Therefore, the Markov chain can easily be expressed with a process in SDL. The SDL model of the SHY model in Figure 3 is illustrated in Figure 5. The SDL model must be complemented with some control signals. Signals must be added to inform the leaf processes when they can make a transition and to acknowledge that a leaf process has made a transition. The signals between the leaves are links between the services of one user (see Section 3.2.2).

The transition in the service process is made randomly, according to the individual profile. The hierarchical profile is updated as the states in the SDL processes are changed.

The SDL model of the usage and the software system communicates with each other with signals. The signals to and from the system are handled in the usage process (see Figure 5).

The logging of the user stimuli and the system's responses is done by placing a process between the system and the usage model. This process writes all the signals, that passes through the process in one or the other direction, on a file. The system is replaced with an oracle as V&V cases are generated (see Figure 6). The V&V cases can be generated and verified before the system is developed. The only input the oracle needs is the requirements specification.

The generation of V&V cases can be made in any language that is desirable. When the file process in Figure 6 logs the stimuli and responses, the process can write in any language on the file if the process knows the syntax of the language. This provides a flexible method of logging software usage.

4.3 Automatic V&V

If the V&V cases are described in the same language as the system, the verification and validation can be done automatically. The system can be executed with the inputs from the V&V cases and the outputs from the system are evaluated by the V&V case (see Figure 7). Whenever the system has a different response for the user stimuli than expected, a failure has occurred. This failure must be examined carefully to see if the failure was in the system's implementation or in the V&V case. As long as the system responds as intended the V&V case has succeeded.

If the V&V cases shall evaluate the system's responses there must be some form of fault detection in the generated V&V case. This fault detection is added to the V&V case when it is generated. The fault detection can be made as a displayed message and the execution can be stopped when the output from the system is not the same as that expected.

The failure data belonging to faults in the system are logged. The data are then fed into a software reliability model, which allows us to accept or reject the software based on an objective evaluation of one of the most important quality attributes. In the early phases of development, the outcome of the analysis is part of the basis for management decisions, hence supporting early planning and control of the software development.

5. Conclusions

Unfortunately, it is infeasible to show all details in the modelling and generation of software usage. However, two issues have been stressed, first the extension of the usage testing concept into other activities of the software life cycle has been discussed briefly. Secondly, the formulation of a usage specification from which V&V cases can be generated semi-automatically. The latter providing the basis for quality control of the software.

The formulation of a general usage V&V process means that the usage testing concept can be applied much earlier in the software life cycle, hence supporting early reliability evaluation and prediction with a user perspective.

The proposed method to model usage is general in the sense that the resulting V&V cases can be input to any activity in the software life cycle allowing for evaluation from a usage point of view. The method is based on modelling usage with a state hierarchy model (the SHY model), which can be easily transformed into an SDL usage model. The latter is particular beneficial if the software is designed using SDL, since it allows for automatic evaluation, but the ideas presented can be used more generally. The SHY model can be transformed to another description technique and V&V cases can be generated in any syntax wanted.

It has been described how V&V cases can be generated semi-automatically by executing the usage specification. The output from the generation forms the input to the V&V process, hence supporting usage evaluation. The strengths of the approach are:

- the hierarchical model, which divide the usage modelling problem into a number of parts, hence making it easier to handle,
- the hierarchical profile, which allows for actually modelling the dynamic behaviour of users of software systems,
- the easiness in which the state hierarchy model is translated into another description technique, in this particular case SDL,
- the opportunity to use available commercial tools instead of developing an in-house tool,

- the generation procedure which enables generation of anticipated usage and also automatic execution of the V&V cases, in particular when using the same description technique for software development and modelling of software usage,
- the ability to actually obtain quality control throughout the software life cycle.

The proposed generation procedure has been evaluated in a number of minor case studies and it is currently being implemented in an industrial project to test the next generation of a case tool for SDL. Thus, the tool used to generate the V&V cases will be verified and validated by the proposed method. Some practical experience is reported in [Runeson95].

In validation and verification the focus must be on the user perspective, since the final evaluation of the software product is made by the user. Therefore, the objective must be to perform user-centered evaluation throughout the life cycle. The proposed method to generate usage V&V cases provides an opportunity to achieve this important objective.

References

[Belina91] Belina, F., Hogrefe, D. and Sarma, A., *SDL with Applications from Protocol Specifications*, Prentice-Hall, London, UK, 1991.

[Mills88] Mills, H. D. and Poore, J. H., “Bringing Software Under Statistical Quality Control”, Quality Progress, pp. 52-55, November 1988.

[Musa87] Musa, J. D., Iannino, A. and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.

[Musa93] Musa, J. D., “Operational Profiles in Software Reliability Engineering”, IEEE Software, pp. 14-32, March 1993.

[Runeson92] Runeson, P., and Wohlin, C., “Usage Modelling: The Basis for Statistical Quality Control”, Proceedings 10th Annual Software Reliability Symposium, pp. 77-84, Denver, Colorado, 1992.

[Runeson95] Runeson, P, Wesslén, A., Brantestam, J. and Sjöstedt, S., “Statistical Usage Testing using SDL”, Accepted for publication, to appear in Proceedings of SDL Forum in Oslo, Norway, 1995.

[Wesslén93] Wesslén, A., “Usage Modelling and Test Case Generation for a Telecommunication System”, Dept. of Communication Systems, Lund, Sweden, Report No. CODEN: LUTEDX(TETS-5180)/1-43/(1993) & Local 30, 1993, Master thesis.

[Whittaker93] Whittaker, J. A. and Poore, J. H., “Markov Analysis of Software Specifications”, ACM Transactions on Software Engineering and Methodology, Vol. 2, No. 1, pp. 93-106, 1993.

[Wohlin92] Wohlin, C., and Runeson, P. “A Method Proposal for Early Software Reliability Estimations”, Proceedings 3rd International Symposium on Software Reliability Engineering, pp. 156-163, Raleigh, North Carolina, 1992.

[Wohlin94a] Wohlin, C., “Evaluation of Software Quality Attributes during Software Design”, Informatica, Vol. 18, No. 1, pp. 55-70, 1994.

[Wohlin94b] Wohlin, C. and Runeson, P., “Certification of Software Components”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 494-499, 1994.