Carl von Ossietzky University of Oldenburg
DFG Graduate School "TrustSoft"
http://trustsoft.uni-oldenburg.de
26111 Oldenburg
Germany

Seminar "Dependability Engineering", Summer term 2005

Operational Profiles for Software Reliability

Heiko Koziolek

9th September 2005

Abstract

Software needs to be tested extensively before it is considered dependable and trustworthy. To guide testing, software developers often use an operational profile, which is a quantitative representation of how a system will be used. By documenting user inputs and their occurrence probabilities in such a profile, it can be ensured that the most used functions of a system are tested the most. Test cases can be generated directly out of an operational profile. Operational profiles are also a necessary part of quality-of-service prediction methods for software architectures, because these models have to include user inputs into their calculations.

This paper outlines how operational profiles can be modelled in principle. Different kinds of usage descriptions of software system have been developed and are summarized in this article.

1 Introduction

Characteristics of dependable software systems are correctness, reliability, availability, performance, security, and privacy. *Reliability* is defined as the probability that a system will perform its intended function during a specified period of time under stated conditions. A common metric to measure reliability is mean-time-between-failure (MTBF), which is the average time to the next failure. To achieve a high MTBF and to be considered a reliable system, software has to be tested extensively.

As testing can almost never assure a complete test coverage, an efficient way of testing has to be found. An *operational profile* is a quantitative representation of how a system will be used [Mus93, MFI+96]. It models how users execute a system, specifically the occurrence probabilities of function calls and the distributions of parameter values. Such a description of the user behaviour can be used to generate test cases and to direct testing to the most used functions. Thus, a practically high reliability of the tested system is achieved.

Descriptions of the user behaviour as in an operational profile can also be used for other purposes than software testing. The performance and correctness of systems can be analysed and systems can efficiently be adopted to specific user groups. If developed early, an operational profile may be used to prioritise the development process, so that more resources are put on the most important operations. It might even be possible to apply an "operational development", meaning that the most-used features of a system are released earlier than other features. An operational profile improves the communication between customers and developers and makes customers think deeper about the features they would like to have and their importance to them.

In the following, a short survey on different operational profiles or usage models for software systems is provided. The differences and limitations of the approaches are described, as well as further applications of usage models.

This paper is organised as follows: Section 2 elaborates on the operational profile approach by Musa by describing the modelling process, listing limitations and introducing extensions to this type of operational profile. Section 3 deals with another form of usage model, namely models based on Markov chains. Additionally, two methods of Markov chain based usage models especially for software components are presented in this section. Section 4 lists applications of operational profiles other than analysing software reliability, and section 5 concludes the paper.

2 Operational Profiles

This section deals with the operational profile model described by John Musa. The steps of creating such an profile are explained, afterwards problems of this approach are outlined. The section concludes with a proposed extension of Musa's model.

2.1 Modelling Operational Profiles

One of the most refereed papers about the development of operational profiles is from John Musa from AT&T Bell Laboratories [Mus93]. His company develops operational profiles to guide the testing of systems. With an operational profile, a system can be tested more efficiently because testing can focus on the operations most used in the field. It is a practical approach to ensure that a system is delivered with a maximized reliability, because the operations most used also have been tested the most.

Musa informally characterises the benefits-to-cost ratio as 10 or greater. In 1993 AT&T had used an operational profile successfully for the testing of a telephone switching service, which significantly reduced the number of problems reported by customers. Hewlett-Packard reorganised its test processes with operational profiles and reduced testing time and cost for a multiprocessor operating system by 50%. Although the effort may vary, Musa estimates the effort for creating

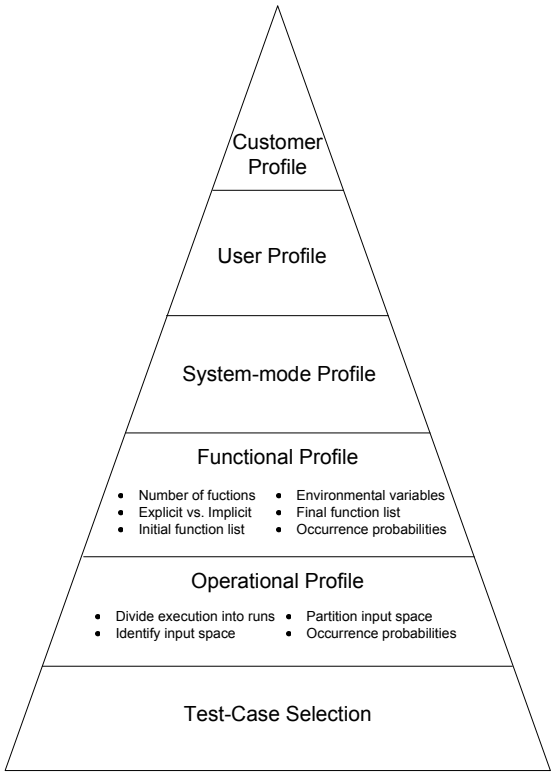

Figure 1: Development Process of an Operational Profile [Mus93]

an operational profile for a typical project with 10 developers, 100000 lines of code and 18 month development time as about one staff month.

The development process of the operational profile as described by Musa successively breaks down system use into five different profiles (Figure 1). A profile is a set of disjoint alternatives with a probability for each item. If service X occurs 90% of the time and service Y occurs 10% of the time the operational profile consists of X,90% and Y,10%. The operational profile is designed by progressively narrowing the focus from customers to operations.

The first four profiles (customer, user, system-mode, functional) are on the design level of a system while the last profile (operational) is on the implementation level and deals with the actually coded operations of a system. For smaller applications it may not be necessary to design each of the first four profiles. For example, if there is only one customer of the software, there is no need to design a customer profile.

Participants of the development process of the profile are system engineers, system designers, test planners, product planners, and marketing professionals. Usage data is either available from similar or older system or has to be estimated, for example based on marketing analysis or on the developers experience. The level of detail of the profiles should mainly be dependent on the expected financial impact, but is – in practice – often defined based on informed engineering judgement. The granularity of the profile can also vary for different parts of the system in relationship to their importance.

In principle, the development of the operational profile is not bound to a specific design methodology or programming language. The design documents may be UML diagrams or result of a structured analysis, and it is possible to create operational profiles for systems programmed object-oriented or imperative.

In the following each of the five profiles is described with more detail.

2.1.1 First Step: Customer Profile

A complete set of customer groups with corresponding occurrence probabilities makes up the *customer profile*. Customers are persons, groups, or institutions that purchase a system. They can but need not to be the users of the system at the same time. Customers in a customer group use the system in the same way. For example, companies with an equal number of employees may use a telephone switching system in the same way because they have the same number of users even though their businesses are different.

Information about the customer profile for new systems must be obtained from marketing by analysing related systems and including the anticipated changes because of the new features in the new system. A simple example for a customer profile would be two customer groups (small and large companies) with respective occurrence probabilities of 70% and 30%.

2.1.2 Second Step: User Profile

A complete set of user groups with corresponding occurrence probabilities makes up the *user profile*. Users are persons, groups, or institutions that use a system. They can, but need not to be, the purchasers of the system at the same time. Users in a user group use the system in the same way. The user profile can be derived by taking the customer profile and determining the user groups for each customer group. Resembling user groups of different customer groups should be combined.

Examples for user groups are system administrators, maintenance users, regular users etc. User groups are usually related to job roles of employees and their numbers might be obtained by counting the job roles for a customer group. The overall occurrence probabilities for user groups can be obtained by multiplying the probabilities for each user group of a customer group with the occurrence probability of that customer group. If user groups are combined over different customer groups, then their probabilities will have to be added. A simple example with the input of the customer profile from above (70% small company (SC), 30% large company (LC)) and 90% regular users (RU) and 10% administrator (AD) in each customer group would result in a user profile of 63% (70% $*$ 90%) SC-RU, 7% SC-AD, 27% LC-RU, and 3% LC-AD.

After the user profile has been developed the development of the subsequent profiles can be delegated to different persons, one user group for each developer.

2.1.3 Third Step: System-mode profile

A complete set of system-modes with corresponding occurrence probabilities makes up the *system-mode profile*. System-modes are sets of functions (design level) or operations (implementation level) that are grouped for a more convenient analysis of the execution behaviour. It is possible to have system-modes that can only be used if no other system-modes are active, but it is also possible to have multiple simultaneous system-modes. The allocation is in the developers' responsibility.

Examples for characteristics of system-modes are user group (administration mode versus regular mode), environment conditions (overload traffic versus normal traffic, initialization versus normal operation), criticality (nuclear power plant controls versus logging functions), user experience (newbie versus expert), or hardware components (functions executed on server 1 versus functions executed on server 2). System-modes can be used to represent the increasing experience of users after introducing a new system.

2.1.4 Fourth Step: Functional Profile

A complete set of functions with corresponding occurrence probabilities makes up the *functional profile*. For Musa a function is a task or part of work of a system as defined during the design. Functional profiles are usually designed during the requirement phases or during early design

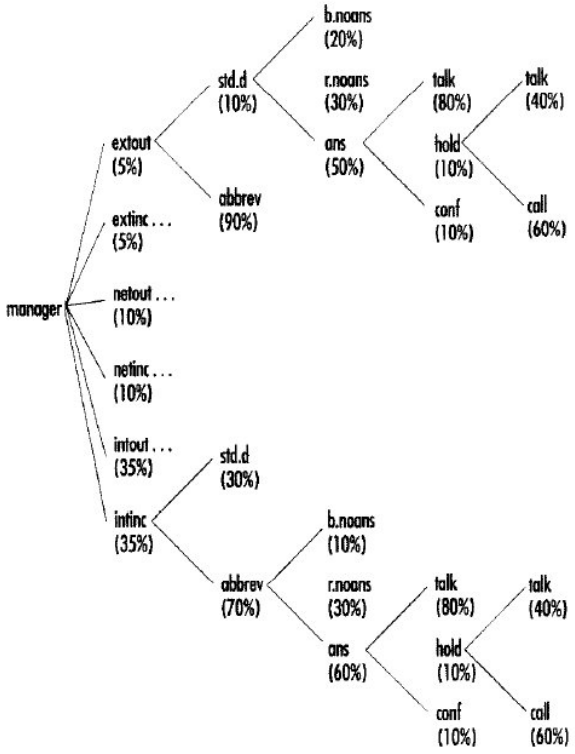

Figure 2: Example of an implicit functional profile [Mus93]

phases. Later, functions have to mapped to operations, which capture a specific behaviour on the implementation level.

Before designing a functional profile it is often helpful to construct a work-flow model capturing the overall processes and the context of the system (i.e. software, hardware, people). To create a functional profile the system modes have to be broken down into single functions. The functional profile is independent of the design methodology and for example might be used for object-oriented or procedural designs.

The *number of functions* in a functional profile is typically between 50 and 300. Criteria for breaking down a system task into two functions are the possibility to develop them with different priorities and the differences in frequency of use. Commands and parameters values are called input variables. Two functions may consist of the same command but different parameters values, because there is a significant difference in the use of value range of the parameters. Input variables that separate functions from each other (in the former case the parameter values) are called key input variables. The granularity of the functional profile depends on the information available during early development stages and the projected amount of costs for a higher precision.

A functional profile may be explicit or implicit. An *explicit profile* includes a cross-product of all key input variables with their possible values and occurrence probabilities, while an *implicit profile* consist of sets for the values of each key input variable with the respective occurrence probabilities. Suppose two key input variables A and B with two possible values for each variable. The explicit profile would be $[(A1, B1), (A1, B2), (A2, B1), (A2, B2)]$, while the implicit profile would be $[A1, A2]$ and $[B1, B2]$. Implicit profiles (consisting of the *sum* of input variables) are smaller but only possible if the key input variables are independent (see example in Figure 2). A disadvantage is that there is no direct selection of input state for the test cases within an implicit profile. Explicit profiles (consisting of the *product* of input variables) are larger, but allow a direct identification of test input states. A combination of an explicit and implicit profile is also possible.

The initial function list contains those functions that are most relevant to the users. In a next step the *environmental variables* such as hardware configurations and traffic loads have to be collected during a brainstorming session of the developers. In Musa's work [Mus93], environmental software such as operating systems or background processes are not considered as environmental variables. After identifying relevant environmental variables, the final function list can be created, which includes the dependencies between key input variables and environmental variables.

The *occurrence probability* for each function in the final function list can be obtained in different ways. If a similar system or even an older release of the software is available, that system can be monitored and the probabilities can be gathered by measurements (e.g. by looking into system logs). If the system under development is new, the probabilities have to be estimated by the developers, which possibly results in an inaccurate functional profile.

2.1.5 Fifth Step: Operational Profile

A complete set of operations with corresponding occurrence probabilities makes up the *operational profile*. Operations, as opposed to functions, are actually implemented tasks of a system, while functions are tasks of a system on the design level. The functions of the functional profile evolve into operations of the system, but the mapping is sometimes not simply one to one. Normally the number of operations is higher than the number of functions, as a single function may be implemented by multiple operations. It is also possible for a set of functions to map to different set of operations. The refinement level of operations is higher, because they include a task with specific input values and value ranges.

To develop the operational profile *runs* are defined, which divide the execution time of a program. Runs are initiated by a specific user intervention or input state and represent an end-to-end user activity. A run type is formed by identical runs. For example the function "change article" in an online-shop may be broken down into two runs, one deleting an article and one adding a new article. Each run type possesses a set of input variables that are used during the run, the so-called input state.

The *input space* of a program is the set of input states that appear during the system's execution and is normally very large, yet finite. The design input space is different from the required input space a program must be tested for, which also contains conditions like heavy traffic or error handling. A list of input states and the corresponding occurrence probabilities has to be defined for an input-state profile. A complete input-state profile normally cannot be defined in practice. Instead a specified input space is defined by listing the involved input variables and their finite number of possible values ignoring the variables with an occurrence probability of zero.

Run types can be grouped into operations and the portion of input space associated with an operation is called a domain. By grouping run types the number of profiles is reduced, which leads to fewer costs but also to less efficient testing. This trade-off has to be considered when designing operational profiles. The partitioning of the input space by identifying domains of operations simplifies the later test generation.

As with the functional profile, there are two way to determine *occurrence probabilities*: by recording input states in the field with similar system or by estimating the values on basis of the occurrence probabilities of the functional profile. For the recording, a general recording tool may be developed which just uses an interface to each application. The estimations should be done by experienced system designers and also reviewed by experienced users.

2.1.6 Sixth Step: Test Selection

With the occurrence probabilities of the operational profile, test cases can be selected efficiently because the most used operations will be tested the most. If an explicit operational profile has

been designed, the test cases can be selected straightforward. If an implicit operational profile has been designed, key input variables and their corresponding values have to be chosen according to their occurrence probabilities, thereby identifying the operations that must be tested. If concurrent system-modes (for example user mode and maintenance mode) occur in the system, it is sensible to also run tests simultaneously to include their interactions in the test. The sequence of operations during the test should be randomized to reduce the bias of the test. Operations that need a special sequence (e.g. a file first has to be opened, then can be read out) should be defined as super-operations.

The number of run categories can be further reduced by only including sequences of two subsequent input variables and excluding sequences of more than two input variables. When conducting regression tests on a system, not only the changed operations should be tested but also all the other operations to reduce the possibility of cross-effects.

2.1.7 Further Issues

A lot of additional research about operational profiles has been conducted. Musa [Mus94] reports, that the error in failure intensity is more than 5 times lower than errors in estimating occurrence probabilities of functions. This implies that developers do not have to put a high effort in precisely determining occurrence probabilities, because the accuracy of these values is not proportionally bound to the failures of the tested systems. Woit [Woi94] specifically describes the specification of operational profiles, test case generation, and reliability estimation for software modules. Avritzer and Weyuker [AW95] present test case generation algorithms for operational profiles and performed load testing for several industrial software systems. Cukic et. al. [CB96] develop another technique for reducing the sensitivity of failure rates to errors in the occurrence probabilities of an operational profile. Bishop [Bis02] shows how reliability bounds can be rescaled in relation to changes in the operational profile. He found out, that it is possible to derive test profiles that are insensitive to a varying operational profile.

2.2 Problems and Limitations

In 2000, Whittaker and Voas [WV00] argued for a rethinking of the operational profile and identified two major problems.

First, using an operational profile emphasises testing the function, which are predicted to be the most used ones. But in practice, users tend not to stay on the path the developers have prepared for them and often use software in an unconventional and unintended way. Functions, for which the developers expected lesser use, might not be tested enough if an operational profile has been used for testing. Thus, using the software in an unintended way decreases reliability rapidly if testing was based on an operational profile. Operational profiles should not only be modelled after the typical user but after all users.

Second, interactions with the software, which are not initiated directly by the user, are not explicitly modelled by an operational profile. Following Musa [Mus93], operational profiles contain a small number of single environmental variables, which represent an oversimplified modelling of the influences on the software. Not only the user creates input to the software, but also the operating system, for example if it signals for the use of resources. Software does not executed isolated on a computer, but other applications usually are running in the background competing for resources. In fact, most parts of the software do not interact with humans, but with device drivers and operating system APIs. Furthermore, humans normally only interact with input device drivers and not with the software itself. The configurations of hardware devices and of other software applications running on the same system influence the behaviour of the software, but are not captured by the operational profile. The operational profile is incomplete and should include

more informations about its environment, especially the operating system, other applications, and system configurations. An appropriate abstraction level should be kept in mind when modelling the environment, otherwise the operational profile would only be valid for a single machine with a specific configuration.

Voas' ideas for countering the second problem can be found in [Voa00]. For him, an operational profile should be defined as the set of events a software receives plus the set of inputs generated by external hardware and software that the software is expected to interact with. To collect the second set of inputs, he suggests to monitor the systems of pre-qualified users, who use the software that shall be tested. For this approach, a prototype or older release of the software has to be available. The software is extended with automated processes that collect usage informations on the computers of the users, of course only with the users' consent. For example data about hardware and software configurations might be obtained from the registry on Windows systems.

To ensure anonymity and privacy of the users participating in such a data collection, Voas proposes the establishment of a middleman organization called Data Collection and Dissemination Lab (DCDL). Not the software developer, but only the DCDL would directly receive user informations and only in an encrypted form. The DCDL would anonymise the data and filter out faulty and unusable data. Additionally, it would ensure that the population of users participating in the test was representative. The resulting data would then be sent to the software developers, who could test the software more extensively, because they then would have a clearer picture in which environments the software will be executed.

2.3 An Extended Operational Profile

Recently, Gittens [Git04, GLB04] tried to solve some of the operational profile's problems like the missing inclusion of the software environment and developed several extensions to the classical approach. This extended operational profile consists of a process profile, a structural profile and a data profile.

- **Process Profile:** Captures processes and their frequencies of a typical usage of the software and is basically the same as Musa's operational profile
- **Structural Profile:** This profile on one hand tries to characterise the data structures of the application and its configuration. On the other hand the profile includes a description of the software and hardware environment of the software.

  The data structures of the application are characterised by so-called *measurable quantities*. Usually they are numerical numbers for the size of a data structure. For example, measurable quantities for a two-dimensional array would be the number of rows and the number of columns. Measurable quantities may change with different configurations of the software or over the course of time. The term data structure does not only refer to arrays, trees or linked lists here. Furthermore, complexer structures like Abstract Data Types (ADT) or modules can also be described with measurable quantities. It is for example also possible to characterise web pages by the number of text fields, buttons, frames etc. Which measurable quantities should be included into the operational profile is the developer's choice. After they are defined, the quantities are recorded by running instances of the software on different systems. Statistics like mean values, median or standard deviations can then be derived from the collected data.

  Some data structures might also be characterised by a fixed number of states, so-called *categorical quantities* they are operating in. For example, a data structure with an overflow flag, which may be set to ON, OFF, and PENDING, has this flag as an categorical quantity

with three associated values. The frequencies of occurrences of the different states can be recorded.

Additionally, the structural profile includes a vector of variables characterising the *hardware environment* and a vector of variables characterising the *software environment*. The authors have applied the extended operational profile in an industrial case study, but do not reveal the concrete values of the hardware and software characterising variables to ensure the privacy of the software vendor's testing and user environment.

In conclusion, the structural profile consists of measurable quantities with statistical values, categorical quantities, and hardware/software characteristics.

- **Data Profile:** This profile is not concerned with the structure of data, but with the actual values variables can be assigned to. A data profile for a database could contain the most occurring data types, the size of single table fields, and the value ranges of table columns. As the number of possibilities for values is normally almost infinite, a high-level view of the data has to be developed, which is the data profile. Values are always recorded for one instance of the software. For each instance, the data profile consists of a number of variables from one particular data type, the value ranges for each data type and the largest data length for each data type from the perspective of the user. These measures are taken from the concepts of boundary value analysis in black-box testing.

As it is difficult and time-consuming to obtain all of the data needed for such an extended operational profile manually, they authors have developed a toolkit assisting designers. As noted before, the authors applied their approach on an industrial case study and, using their toolkit, needed eight person hours to collect the necessary data.

3 Usage Models Based on Markov Chains

Operational profiles as in Musa's approach [Mus93] do not explicitly consider the dependencies between different inputs to a software system. An operational profile is structured like a tree, with operation calls as the leafs and probabilities on the branches. Not included are relationships between consecutive calls, also known as protocols. For example if a specific call always requires a certain predecessor (e.g. `openFile()` has to be called before `writeFile()`), this can not be expressed explicitly by the operational profile.

3.1 Markov Chain Usage Model

Whittaker et. al. [WP93] first proposed using Markov-chains for *modelling sequences* of inputs to a software system. Like Musa [Mus93] they describe usage for the purpose of generating test cases and to guide software testing statistically. Ultimately, the reliability of a system shall be improved by extensively testing the most-used functions. The software system is viewed as a black box, which receives stimuli from the outside. In particular sequences of stimuli representing traces of the software execution are of interests to the authors. These sequences directly represent test cases and can be used in a random experiment, which is conducted for the statistical software testing. To describe the test cases, a set of random variables is used, which models the complete set of sequences the user can execute.

A sequence of events can be expressed as a stochastic process. In this approach finite state, discrete parameter *Markov chains* are used to model the sequences. The states of the Markov chain represent inputs to the software system, while the arcs imply an ordering of the inputs and are annotated with probabilities. The Markov property adds that for each arc, the next state is

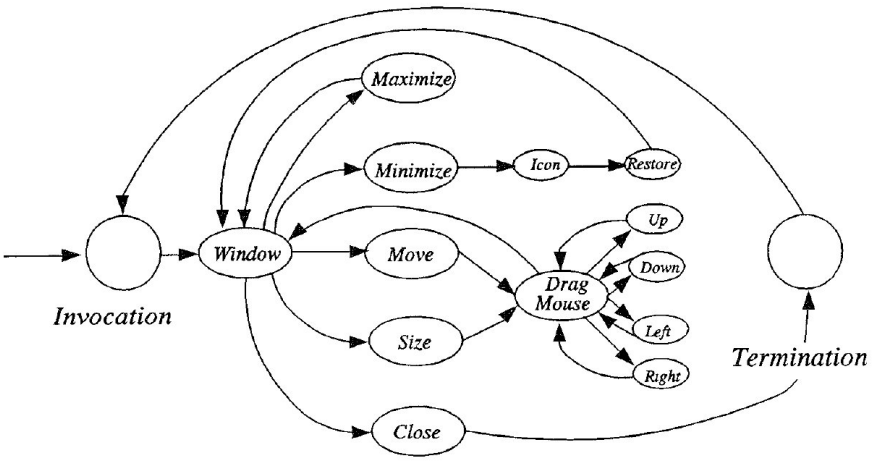

Figure 3: Exemplary Markov model after structural phase [WP93]

independent of all past states given the present state. An advantage of using Markov chains is the rich body of theory with analytical results and computational algorithms.

The *development process* of the Markov chain is divided into two steps: the structural phase and the statistical phase. During the *structural phase*, a state is created for every possible action the system is able to receive. Arcs are added to connect consecutive actions. The design of the structure is creative process, as there is no algorithm to support this phase. An example for the result of the structural phase for the manipulation of a window in a graphical user interface can be found in Figure 3.

After the structure of the Markov chain has been established, probabilities are assigned to the arcs during the *statistical phase*. There are three methods to do this:

- **Uninformed approach:** If no information about the expected probabilities is present, this approach is the only possibility. The exit arcs of each state are assigned with a uniform probability distribution. This results in a single unique model, but is not a close resemblance of the actual probabilities.

- **Informed approach:** If a prototype or older release of the software system is available, the informed approach can be used. User behaviour can be monitored, and the measured frequency counts of taking each arc in the Markov chain can be converted into transition probabilities. This approach may lead to different models depending on the monitoring data.

- **Intended approach:** If no similar system is available, at least the experienced designer is often able estimate the expected transition frequencies with a careful and reasonable analysis of the user behaviour. This is the intended approach, which also results in different Markov chains depending on the designer.

The corresponding probabilities to the window example from Figure 3, which have been determined during the statistical phase, can be seen in Figure 4.

Using Markov chains yields the advantage, that several analytical descriptions of the test cases can be made based on the model. For example the number of states necessary before reaching a certain state or the mean first passage time can be calculated out of Markov chains.

They authors used their approach on a simple spreadsheet program, for which the identified 90 states and over 200 arcs. Additionally, they created a usage model for the IBM DB2 database, which consisted of more than 2000 states, yet the models were still analytically tractable. It has to be kept in mind that even small software systems can have a large input space, so that a Markov

Table II. Statistical Phase—Assigning the Transition Probabilities

From-State	To-State	Frequency	Probability
Invocation	*Window*	6	1
Window	*Maximize*	1	1/12
Window	*Minimize*	1	1/12
Window	*Move*	2	1/6
Window	*Size*	2	1/6
Window	*Close*	6	1/2
Maximize	*Window*	1	1
Minimize	*Icon*	1	1
Icon	*Restore*	1	1
Restore	*Window*	1	1
Move	*Drag Mouse*	2	1
Size	*Drag Mouse*	2	1
Drag Mouse	*Window*	4	4/15
Drag Mouse	*Up*	1	1/15
Drag Mouse	*Down*	5	1/3
Drag Mouse	*Left*	3	1/5
Drag Mouse	*Right*	2	2/15
Up	*Drag Mouse*	1	1
Down	*Drag Mouse*	5	1
Left	*Drag Mouse*	3	1
Right	*Drag Mouse*	2	1
Close	*Termination*	6	1
Termination	*Invocation*	-	1

Captured or hypothesized sequences:

1. <*Invocation*><*Window*><*Maximize*><*Window*><*Close*><*Termination*>
2. <*Invocation*><*Window*><*Minimize*><*Icon*><*Restore*><*Window*><*Close*><*Termination*>
3. <*Invocation*><*Window*><*Move*><*Drag Mouse*><*Down*><*Drag-Mouse*><*Right*><*Drag Mouse*><*Down*><*Drag Mouse*><*Window*><*Close*><*Termination*>
4. <*Invocation*><*Window*><*Size*><*Drag Mouse*><*Left*><*Drag-Mouse*><*Up*><*Drag Mouse*><*Left*><*Drag Mouse*><*Window*><*Close*><*Termination*>
5. <*Invocation*><*Window*><*Move*><*Drag Mouse*><*Down*><*Drag-Mouse*><*Left*><*Drag Mouse*><*Down*><*Drag Mouse*><*Window*><*Close*><*Termination*>
6. <*Invocation*><*Window*><*Size*><*Drag Mouse*><*Down*><*Drag-Mouse*><*Right*><*Drag Mouse*><*Window*><*Close*><*Termination*>

Figure 4: Exemplary probabilities for Markov chain after statistical phase [WP93]

chain with many states has to be created. But even then, the authors assume a manageable computational effort for the analysis of these model.

3.2 Hierarchical Markov Chain Usage Model

Wohlin and Runeson [WR94] also use Markov chains for usage modelling, specifically for the reliability engineering of software components. Their usage model is divided into an usage structure containing possible sequences of service calls and a usage profile containing probabilities of control flow branches. The overall aim of this work is to provide a basis for the certification of components in terms of reliability measures for certain usage models. The approach of certification consists of 5 steps:

1. Modelling of the usage structure
2. Modelling of the usage profile
3. Generation of test cases out of the usage model
4. Execution of test cases and collection of failure data
5. Certification of reliability and prediction of future reliability

The usage models by Wohlin and Runeson describe the user behaviour for a complete system as well as for individual components from an external view. Users may be either human beings or other components. Because the usage model is divided into usage structure and usage profile the model can be easily reused. For example by changing the probabilities of the profile while retaining the usage structure the usage model can be adapted for a different system context.

A hierarchical Markov model, the so-called *state hierarchy model* (SHY) is used for the representation of the usage model. A disadvantage of using Markov models is the possible exponential growth of the state space and thus the intractability of these models, if they are applied to complex software systems. To cope with the state space explosion the SHY models consists of five levels, and the behaviour of single services can be described separately before being composed into one big model (Figure 5).

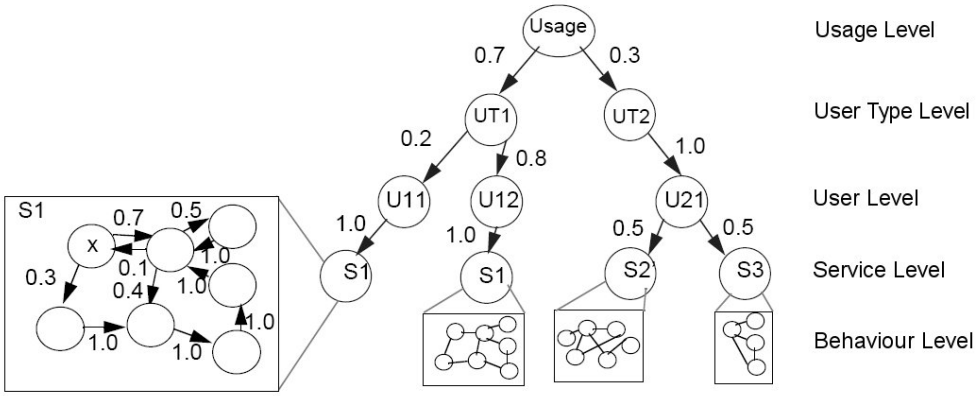

Figure 5: State hierarchy model [WR94]

The *usage structure* can be divided into different user types (for example regular users and administration users). From the user type level the behaviour of single users can be modelled. For each user a number of services of a components is being described, and for each service the individual behaviour is being described as a Markov model on the lowest level of the SHY model. The interaction of different services can be modelled on this level be creating links from one Markov model to another.

After modelling the usage structure, each branch in the usage model is assigned with a probability, thereby adding the *usage profile*. The values for the probabilities must be derived from similar systems including expected changes, from the experience of the developers or from the expected usage of the system as described in the system's specifications. Probabilities are normally static, but also can be dynamic, expressing the fact that some events are more probable under certain conditions. Because it may be impossible to determine usage profiles reflecting the exact execution of a components, it is more important to find reasonable probability relations.

Test cases can be generated by going top-down through the SHY model randomly selecting users types, single users, services and the corresponding Markov models. After additionally generating input parameters, the stimulus of a Markov model on the behaviour level can then be added to a test script. This procedure can be performed iteratively to gain a high coverage of the usage model.

The *certification* is carried out by proposing a hypothesis, which states if a specific MTBF (mean-time between failure) requirement can be met with a specific degree of confidence. The goal of testing the component is to find out whether the hypothesis can be accepted or rejected. If the hypothesis is neither accepted nor rejected during the testing process, testing has to continue until the needed degree of confidence is reached. For the certification the failure number (r) is plotted against the normalized failure time (t) (Figure 6). Normalizing of the failure time is done by dividing the failure time by the required MTBF. Testing is performed as long as the measured data points fall into the "continue" region and terminated, if the data points fall into the "accept" or "reject" region. More details about the hypothesis certification can be found in [MIO87].

New components can be certified for a particular usage profile with specific reliability measures. The reliability measures can be stored into a component repository with the component, so that third-party-users have a guiding value when assessing the component for possible use in their architecture. However, the certified measures may not be reused blindly, because the usage profile the component has been specified against is arbitrary and normally cannot be replicated exactly by a potential user of the component. The component user has to take his special usage characteristics into account when assessing the true reliability of the component. For example, the component user can change the probabilities of the usage profile and re-certify the component for his usage context. By certifying components against more and more usage profiles, the trust into reusing

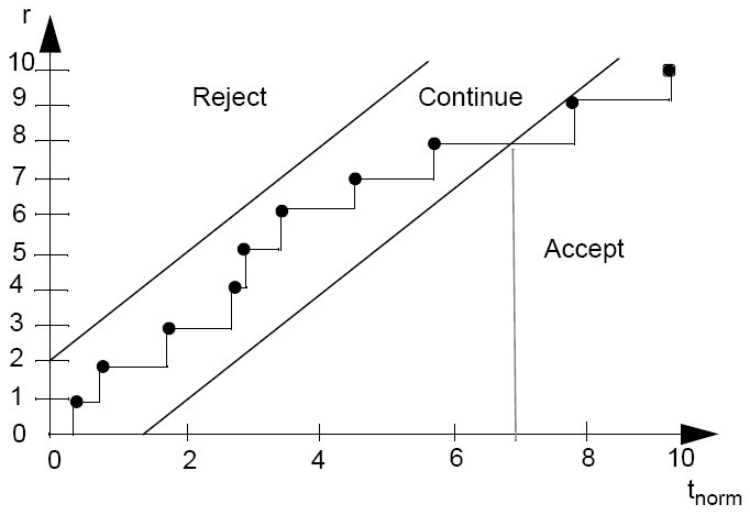

Figure 6: Control chart for hypothesis certification of the reliability [WR94]

these components will be increased, because the components have been tested for a large number of usage contexts.

3.3 Probabilistic Statechart Usage Model

A recent approach specifically for usage modelling of software components built on the work by Whittaker and Poore, and used probabilistic statecharts (Figure 7) to describe the usage structure and profile [SCS04]. With the use of statecharts, the authors hope to overcome the state explosion problem of Markov chains, which often become intractable for larger system. Yet they do not explicitly show the advantages of this modelling formalism. This proposal considers dependencies between the parameters of consecutive calls.

The development of the probabilistic statecharts consists of four steps. First, relevant information is gathered including descriptions of the interfaces of the components as well as traces of usage data from a prototype or from simulation. Assumptions are made about the expected use, where no measurements are available. Afterwards, the structure of the statechart is modelled. This can be achieved in a top-down manner, going through the usage traces, grouping related operations into sequences, and designing statecharts for these groups. It can also be done in a bottom-up manner, first defining states for every operation, and then adding transitions branches starting from the initial state.

In a third step, a transition matrix is constructed containing the probabilities for the transitions of every state to every other state. For this purpose, frequencies of calls from the traces are translated into probabilities. The fourth step consists of a parameter analysis. By looking at the interfaces of the component, the parameter types can be determined. Constraints for individual parameters are described as well as relationships between different parameters. For example the output parameter of one function call might be the input parameter for the next call. These descriptions are documented textually.

With the completed probabilistic statechart test cases can be generated. The authors wrote a Java program for this purpose.

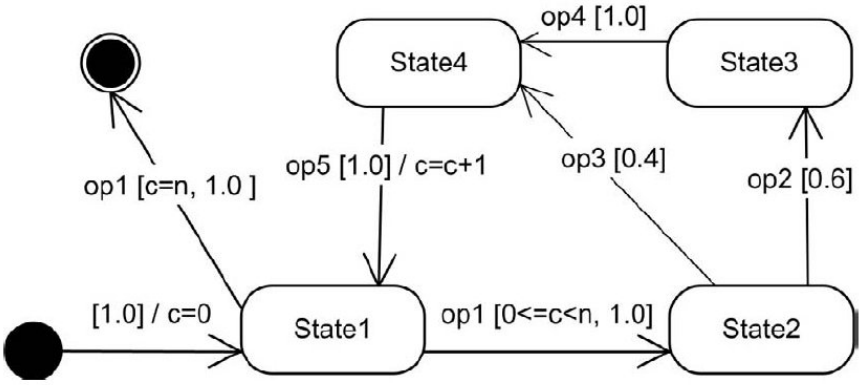

Figure 7: Example for a probabilistic state chart as a usage model [SCS04]

4 Other Applications of Operational Profiles

Originally, the primary aim of designing operational profile was the generation of test cases, the guiding of development and testing to the most-used functions of a system, and the reliability analysis of software systems. However, operational profiles are useful for other purposes as well, e.g., performance and reliability prediction, detection of redundant code and web usage mining.

Performance Prediction There are a large number of performance prediction methods for software architectures that instrument operational profiles for their calculations and analyses. A comprehensive survey of such methods can be found in [BDIS04]. These methods try to analyse software architectures before they are actually implemented and take models of the software as inputs. Nowadays, UML diagrams are the de-facto standard for documenting designs, and there is a special UML profile (UML Profile for Schedulability, Performance, and Time [OMG03]) to include performance related annotations like computing times or rates of incoming requests into UML models. In fact, the operational profile of the proposed architecture can be specified coarse-grainly with this profile. For example, it is possible to annotate single use cases with occurrence probabilities and input frequencies. Performance prediction methods have to take into account these annotations because the operational profile is a major influencing factor to the performance of a system. For example, if a certain method is most frequently used with large-sized parameters instead if small-sized parameters, the average response times of this method is expected to be rather long.

Most performance prediction methods either transform UML diagrams into performance models or directly use such models. Formalisms like queueing networks, stochastic petri-nets, stochastic process-algebras and markov-models are most common to describe performance models. These models need occurrence rates of incoming requests as well as transition probabilities between different states of the system as an input for their evaluation. These informations are part of the operational profile.

An approach specifically for the performance prediction of component-based system can be found in [BM04]. To analyse the performance of a component-based architecture, an operational profile is developed for the whole system in this method. Hamlet et. al. [HMW04] partition their operational profile for software components into subdomains and use a finite vector approximation of these subdomains, because the exact operational profile is never available in practice. They also describe how requests to these subdomains fall into the subdomains of following connected components. With these informations, they are able to calculate the expected performance of a complete component-based architecture.

Detection of Redundant Code Alzamil presents an approach for identifying redundant statements in source code with the help of an operational profile [Alz04]. Redundant statements are statements that might be executed, but removing them would not alter the functionality of the program. Whether a statement can be considered redundant partially depends on the operational profile. If users executed the software in a specific way, it might happen that certain statements are not used in a way that would change the program's output. For example, an algorithm identifying the minimum value of an array of integer-variables does not have to get the value of each element of the array, if the users always call this algorithm with a sorted array and the minimum value is always the first value. The reason for eliminating such redundant statements is the improvement of the performance of the programs.

The author conducted a case study and tested multiple programs looking for redundant statements. At first, random inputs were used to test the software, then a manually generated operational profile was used. In 80% of the cases using the operational profile yielded a significant higher number of found redundant code statement. Thus, the performance of the respective programs could be improved more effectively with the help of operational profiles.

Web Usage Mining A completely different domain involving the analysis of usage data is web usage mining (for example in [MDLN02]). These approaches try to identify patterns in the user behaviour of web applications. The aim is the personalisation of web site contents. For example, an online shop may be able to make recommendations for products relevant to the user based on the products he or she viewed before. The methods shall be suited even for anonymous users not registered to web applications. Patterns like association rules, sequences, and clusters of user sessions are identified with data mining techniques, afterwards aggregate usage profiles are derived from these patterns.

5 Conclusions

Several approaches for specifying user behaviour have been presented in this survey paper. The classical method of developing operational profiles by Musa has been used extensively for software reliability engineering. After designing different levels of profiles, finally an operational profile on the implementation level can be specified, from which test cases can be selected. Testing the most used functions ensures a high software reliability. Problems of the approach, namely the negligence of the hardware/software environment and the focus on ideal users have been explained as well as possible extensions to solve these problems.

Another class of usage models are based on Markov chains and can also model dependencies between consecutive calls to a software system. In this class, a state hierarchy model has been developed, furthermore probabilistic statecharts have been used to model user behaviour.

Still missing in most models is a proper treatment of parameter values. Probability function could be used to model the value ranges of input parameters. The dependencies between the parameter values of consecutive calls could be modelled explicitly. Apart from Hamlet's work there is no approach modelling the transformation of operational profiles between multiple software components. Executing one component with a specific operational profile does lead to another operational profile on the components that the first component is using to provide its services. To ensure reliability and for sensible test case generation these transformations need to be modelled explicitly. Including the software environment into the operational profile has been tried by Gittens, yet the approach is limited in expressiveness.

Apart from reliability engineering, operational profiles and usage models can be used for other purposes. In this paper, the examples of performance prediction, redundant code detection, and web usage mining can be found.

References

[Alz04] Alzamil, Z.: Application of the operational profile in software performance analysis. In: *WOSP '04: Proceedings of the fourth international workshop on Software and performance*, New York, NY, USA: ACM Press, 2004, ISBN 1-58113-673-0, pp. 64–68, doi:http://doi.acm.org/10.1145/974044.974053

[AW95] Avritzer, A.; Weyuker, E. J.: The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software. In: *IEEE Trans. Softw. Eng.* 21 (1995), № 9, pp. 705–716, ISSN 0098-5589, doi:http://dx.doi.org/10.1109/32.464549

[BDIS04] Balsamo, S.; DiMarco, A.; Inverardi, P.; Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. In: *IEEE Transactions on Software Engineering* 30 (2004), № 5, pp. 295–310

[Bis02] Bishop, P. G.: Rescaling reliability bounds for a new operational profile. In: *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, New York, NY, USA: ACM Press, 2002, ISBN 1-58113-562-9, pp. 180–190, doi:http://doi.acm.org/10.1145/566172.566201

[BM04] Bertolino, A.; Mirandola, R.: CB-SPE Tool: Putting Component-Based Performance Engineering into Practice. In: *Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings*, Springer, 2004, vol. 3054 of *Lecture Notes in Computer Science*, ISBN 3-540-21998-6, pp. 233–248

[CB96] Cukic, B.; Bastani, F. B.: On reducing the sensitivity of software reliability to variations in the operational profile. In: *ISSRE '96: Proceedings of the The Seventh International Symposium on Software Reliability Engineering (ISSRE '96)*, Washington, DC, USA: IEEE Computer Society, 1996, ISBN 0-8186-7707-4, p. 45

[Git04] Gittens, M.: *The Extended Operational Profile Model for Usage-Based Software Testing*. PhD thesis, Faculty of Graduate Studies, University of Western Ontario, 2004

[GLB04] Gittens, M.; Lutfiyya, H.; Bauer, M.: An Extended Operational Profile Model. In: *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, Washington, DC, USA: IEEE Computer Society, 2004, ISBN 0-7695-2215-7, pp. 314–325, doi:http://dx.doi.org/10.1109/ISSRE.2004.8

[HMW04] Hamlet, D.; Mason, D.; Woit, D.: *Properties of Software Systems Synthesized from Components*, World Scientific Publishing Company, vol. 1 of *Series on Component-Based Software Development*. March 2004, pp. 129–159

[MDLN02] Mobasher, B.; Dai, H.; Luo, T.; Nakagawa, M.: Discovery and Evaluation of Aggregate Usage Profiles for Web Personalization. In: *Data Min. Knowl. Discov.* 6 (2002), № 1, pp. 61–82, ISSN 1384-5810, doi:http://dx.doi.org/10.1023/A:1013232803866

[MFI+96] Musa, J.; Fuoco, G.; Irving, N.; Kropfl, D.; Juhlin, B.: *The Operational Profile*, IEEE Computer Society Press and McGraw-Hill Book Company. 1996, pp. 167–216

[MIO87] Musa, J. D.; Iannino, A.; Okumoto, K.: *Software reliability: measurement, prediction, application*. New York, NY, USA: McGraw-Hill, Inc., 1987, ISBN 0-07-044093-X

[Mus93] Musa, J. D.: Operational Profiles in Software-Reliability Engineering. In: *IEEE Software* 10 (1993), № 2, pp. 14–32

[Mus94] —— Sensitivity of field failure intensity to operational profile errors. In: *Proceedings., 5th International Symposium on Software Reliability Engineering*, 1994, pp. 334–337

[OMG03] OMG, O. M. G.: UML Profile for Schedulability, Performance and Time. `http://www.omg.org/cgi-bin/doc?formal/2003-09-01`, 2003

[SCS04] Shukla, R.; Carrington, D.; Strooper, P.: Systematic Operational Profile Development for Software Components. In: *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, Washington, DC, USA: IEEE Computer Society, 2004, ISBN 0-7695-2245-9, pp. 528–537, doi:http://dx.doi.org/10.1109/APSEC.2004.95

[Voa00] Voas, J.: Will the Real Operational Profile Please Stand Up? In: *IEEE Softw.* 17 (2000), № 2, pp. 87–89, ISSN 0740-7459

[Woi94] Woit, D.: *Operational Profile Specification, Test Case Generation, and Reliability Estimation for Modules*. PhD thesis, Queen's University, Kingston, Ontario, Canada, 1994

[WP93] Whittaker, J. A.; Poore, J. H.: Markov analysis of software specifications. In: *ACM Trans. Softw. Eng. Methodol.* 2 (1993), № 1, pp. 93–106, ISSN 1049-331X, doi:http://doi.acm.org/10.1145/151299.151326

[WR94] Wohlin, C.; Runeson, P.: Certification of Software Components. In: *IEEE Trans. Softw. Eng.* 20 (1994), № 6, pp. 494–499, ISSN 0098-5589, doi:http://dx.doi.org/10.1109/32.295896

[WV00] Whittaker, J. A.; Voas, J.: Toward a More Reliable Theory of Software Reliability. In: *Computer* 33 (2000), № 12, pp. 36–42, ISSN 0018-9162, doi:http://dx.doi.org/10.1109/2.889091