Automated Test Distribution Framework for Service Providing Systems

Mirko Caspar*, Matthias Vodel*, and Wolfram Hardt*

Abstract

Testing is still the only way to verify complex systems. As different technologies for testing small components of a system are established, e.g. unit tests, no common methodologies or approaches exists for testing the whole systems. Reasons for this purpose is the diverseness of the systems to be tested. But during development, especially the testing of whole systems, composed of distributed and communicating subsystems, is very important to find errors or weaknesses in the designs. Since the complexity of such systems causes an immens amount of testcases, an automated process is preferable.

In this paper a general framework for automated system testing is introduced. It can be used for functional blackbox testing of any kind of systems which provide services that can be consumed by any kind of clients. Main issue is the automatical generation of an amount of service requests based on an abstract test-scenario. Since these requests has to be executed by the clients, a test-infrastructure is part of the framework and allows to communicate remotely with a dynamical number of distributed clients. Information about success of all requests are collected, analysed and can be used to influence the test-scenario. So it is possible, to provoke situations which indicates problems. The usage of this new framework allows to generate tests, which can help developers to find errors or weaknesses in the designed system. Furthermore it can be used to maintain or benchmark a system.

1. Introduction

"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."- who wants to controvert to Edsger W. Dijkstra statement [3] from 1972? We guess - or hope - no one. Unfortunately, systems became much more complex in the last 25 years. And unfortunately, we don't have an alternative to testing if we want to verify complex systems. Due to technological and economical reasons the question is important: how can as much bugs as possible be found by testing?

It's hard enough to find relevant testcases for simple stand alone software. But due to different technological developments in the past years, especially distributed or networked systems growed. These systems are characterised by the participation of many - mostly a dynamical number of - subsystems, which interact in a defined way. A typical and widely spread example for such systems are client/server infrastructures, where a server provides some services, which are consumed by a big number of clients. These clients communicate via a network with the server. If the whole system should be testes, testcases for the clients, the server and the network infrastructure have to be created. But what is about the interaction of these devices? And if the system gets more complex, even the server itself may be distributed over many interacting subsystems. The network can be a hierarchical one containing a lot of embedded systems which interconnect different devices.

Since this system complexity can not be handled in one big test, hierarchical test strategies are applied in practice - following the V-Model of software engineering (see fig. 1). During the development of single components, the particular functionality is checked by unit tests. These tests can be static or dynamic white-box tests, because the developer knows about the weak points of his implementation and can survey these by applicable testcases.

If a common degree of error freeness for all components is achieved, the subsystems are combined to the whole system and can be checked under laboratory conditions. This means, that the environmental conditions of the system can be controlled. By execution of black-box tests different functional parameters can be checked in respect to the specification. It is possible to use special test software or hardware. However, it is not possible to emulate all dependencies for these tests in laboratory. A simple example is the mobility of clients or the individual handling of a user.

To test these parameters, a field test follows. Hereby, the whole system is implemented in a real world runtime environment. Mostly, test users are recruited, who shall create the charging to the system-under-test, e.g. by using the mobile clients. So, not only these parameters can be tested, which are hardly to be emulated in laboratory condition, but also a kind of not planed fortuity is integrated in the test. But it is a disadvantage of this kind of test, that a developer has no opportunity to influence the testcases created by the test users, because he is not able to coordinate the actions of them. So it is not possible to provoke special testcases or situations or to get any kind of reproducibility.

Looking for answers to the question of finding as many bugs as possible, it seems to be a interesting approach, to combine the advantages of laboratory and field system tests. A test mechanism would emerge, where developers and test engineers get the possibility to get testcases from a random

* *Chair of Computer Engineering, Dept. of Computer Science, University of Chemnitz, Chemnitz, Germany*

behaviour of a test user and, however, can influence the parameters of the test to provoke special system states.

Initiated by the requests of an industrial partner, in this paper a test framework is presented, which realises the announced advantages for heterogeneous service providing systems of almost any complexity. It allows to use distributed clients to execute testcases remotely. The possibility of test automation is also opened by introducing abstract service loads depending on time, which will be decomposed to testcases. To achieve this functionalities, test routines are implemented on the service consuming clients. These routines are controlled and parameterised by a central test server. Beside the possibilities of generating testcases for debugging, the system will cover a wider range of test aspects, like load, stress or performance testing.

Following this introduction, section 2 will give an overview of research and technologies in the area of testing complex systems. Section 3 will introduce a new approach for automated test distribution and discuss the requirements. The details of the resulting framework are shown in section 4. Advantages and disadvantages of this new approach are discussed in section 5 before this paper is closed by a conclusion and outlook.

2. Related Work

A special and much more simple case of remote testing is the idea of remote diagnostics and management, which is relative old. A lot of work on different areas has been done here. Especially network providers are interested in a possibility to check the operation of user devices. In [2] a protocol can be found, which allows to check the transmission of data in an ISDN network. In [10] an approach of a failure management for next generation of cellular networks is presented, where an algorithm tries to find correlation between errors. A very formal approach is presented in [13]. The authors introduce an in-system diagnosis and define a information flow model to describe the amount of information, which can be concluded from different tests. However, these approaches aim to scan for reasons of errors during system runtime and are not able to generate testcases for complex, service providing systems.

Beside, a lot of literature is available about testing of software. In software engineering, testing is considered as a technology to improve the quality and known to be a important part of the development costs [9]. So a lot of economic work is done to improve efficiency of testing, like planning and controlling test teams, selecting test tools and so on [5][6]. A widely spread and common structure for the development and test process is hereby seen in the V-model which is shown in fig. 1. In opposite to Royce's Waterfall Model [11], a hierarchical test process is set in correlation to the different steps of specification and implementation.

For the level of unit tests a lot of methodologies and technologies where developed and implemented. One of the most famous is the JUnit implementation, which allows semi automated test generation for Java programs. The unit test process is even standardised by the IEEE [1]. On the other hand, common models or processes for higher level testing, like system tests, are not available.

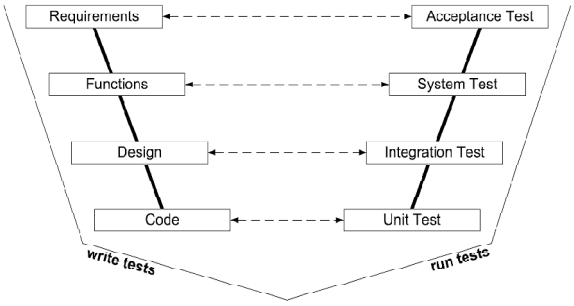

Fig. 1. The V-model of Software Engineering [6]

Due to the high costs of producing an executable hardware it is essential for development of hardware to find as much design bugs as possible before the production of prototypes starts. So the verification of hardware is traditionally done by simulation and newly by formal proofing methods. In the latter case, a formal model of the system is proven against a formal model of the specification [4]. So it is possible, to show the "absence of bugs".

In addition to these general methods of testing hard- and software, a lot of approaches for testing specialised system types, infrastructures or implementations exists. Especially in the field of testing networks a lot of research work has been done. In [15] a distributed algorithm is proposed, which allows to check the quality of routing protocol implementations.

Beside these publications and academic work, a lot of commercial products exists, which are labeled with "test automation". But most of them are better macro executers, which can repeat GUI clicks or user actions, which were recorded before [6].

3. Distributed Remote Testing

As mentioned in the sections before, there is no common and accepted process or technology for system testing. This may result from the diverseness of the systems to be tested. In this paper a general process and framework for abstract system testing of service providing systems (SPS) is defined.

Hereby, a SPS is a system with well defined interfaces, whereat requests of a dynamical number of service consuming clients is processed and responded. A typical example for such a system is a server, which provides a number of services for requesting clients. Hereby, the abstraction of such systems can be very different. A simple DHCP server is also a SPS like a whole, geographical widely spread cellular network, e.g. UMTS, which provides services like phone calls, sending of sms and file downloads.

To define a common system test process for all this types of systems it is necessary to know what commonly should be tested. Since the task of an SPS is, to provide the defined services for a various number of requesting clients, the following hypothesis will be used as basis for the presented test process:

Hypothesis: The definable generation of a various amount of requests by a various number of clients is an efficient way for system testing of SPS.

In other words, it is assumed that an efficient method to

test a system is to generate load for this system. Hereby, load is a certain amount of service requests. In case of the mentioned SPS this means, that the clients send requests to the SPS and check for correct responses.

Dialogues with industrial partners, who develop such systems, indicates that this proposition is covered by daily practice, especially as one part of the laboratory test and main task of field test (see section 1).

3.1 Requirements

To realise such a test system, some requirements has to be considered. Following the hypothesis, clients sends requests to the system, which has to be tested. Hence, several functions for accessing theses SPS services has to be implemented for each of the clients.

The clients can be logical and geographical distributed. So a technique has to be realised, which allows a central point to control the client test functions. If the availability of clients is dynamic and can change in time, an additional datastructure is necessary, which contains information about the currently available clients and the actual possibility to get a connection to them.

Furthermore, in many cases it is necessary to execute a huge amount of service requests in a short time. So, a manual definition of these test by a test engineer is not possible. It is necessary, that the test engineer describes a test in a much more abstract way, based on the hypothesis. This description has to be used to generate the service requests for the clients automatically.

The presented approach should particulary be suited for black-box testing. So, the responses about the success of the particular service requests are the only information about the results of the test. These feedbacks have to be picked up from the clients and stored at a central point. Furthermore, it is advantageously to use this feedbacks not only for test evaluation by the test engineer or developer but also for an algorithmic analysis. If conspicuities can be found this way, these can be verified by automatically generated tests. An increased test coverage and efficiency of test can be reached by this process.

3.2 Classification

As seen in section 2, a lot of possibilities and realisations of tests exists. So a classification for the presented approach is necessary before discussing details of it.

First of all, the approach allows testing of abstract systems. It is not differed between hardware and software or network and server. All subsystems, which are necessary to provide services and to communicate with consuming clients, are included in the test. Therefore, many domains of systems can be tested in one step, like complex distributed software running on servers and connected through a network, which is maintained by embedded systems.

As mentioned in the V-Model (see fig. 1), functional blackbox testing is the usual methodology for system tests. The same applies to this approach. Due to the heterogeneous and complex subsystems it is not possible to include details about the implementation in the test strategy. It is also not necessary, since these details has to be checked during phase of unit tests. Main focus of testing SPS is the check of availability of the provided services.

A weakness comes along with the black-box testing. The test engineer is not able to get information about states of the system-under-test. So it is not possible to detect errors within the SPS. The only guaranteed information about executed tests are the remote responses of the test clients, which may include information about the success of the requests, error details and timing information. An analysis of these results can be used to detect possible relations between the availability of different services or abnormal service behaviour. This can afterwards be reviewed by developers, who are looking at traces or repeat the test with same parameters.

4. Framework

The aim of this work is to get an automated test system, which covers most of the possible test scenarios described in the hypothesis from section 3. A specific test system for a particular SPS can than be derived from this general solution. This leads to a system as it is shown in fig. 2.

It is composed of 4 subsystems. The test-infrastructure controls the connection to the test-clients and sends test-atoms to them. The atoms contains information about the type of request, which has to be run on the SPS by the clients. Since it may be necessary to create a huge amount of such atoms in a short time, a test-automation system generates this atoms automatically based on an abstract test-scenario. To improve the quality of tests, a test-analysis systems checks the testatom responses for noticeable problems and may influence the test-scenarios to provoke appropriate situations.

The system is discussed in detail in the following sections.

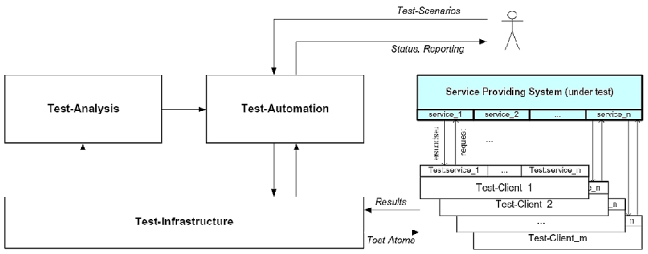

Fig. 2. Overview over the whole system composed of test-clients, testinfrastructure, test-automation and test-analysis

4.1 Test Clients

The test clients are the service consumers which sends requests to the SPS. This can - but don't have to - be user clients in the system too. For example, it can be employee workstations in a network, which have access to a file server.

These test clients have to implement test functions, which can be accessed and parameterised by the test-server. For each class of service, which should be tested, an appropriate test function implementation is necessary. After executing a test, a report about the success of the service request has to be sent to test server.

If possible, the test functions should use the same libraries and resources like the future application of a client, for example the TCP/IP stack. So it is assured that the same detailed

parameters used for the requests to the SPS.

4.2 Test-Infrastructure

The test-infrastructure (shown in fig. 3) manages the remote access of available test clients. Beside the test-server, a communication medium belongs to the infrastructure which allows communication to the clients. In general this is some kind of network. In special cases, this medium can be part of the SPS, like it is in the example of a cellular network. In this case, the communication between test-server and test client is already a service request, which is focus of the test.

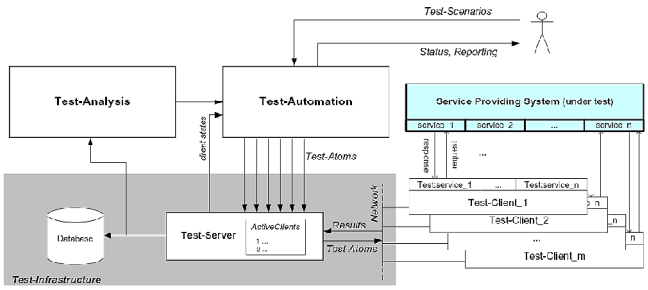

Fig. 3. Detailed overview of the test-infrastructure component

To manage the clients, the test server maintains a list, where all clients with available connection to the test-server have to register. Connection parameters, like an IP address, are stored. So the server is able to contact the clients and transmit the test-atoms, which are defined as:

Definition 1: A test-atom is a tuple of sets which contains information about the test function, the client to run this function and the necessary parameters to start this test.

The tuple of parameters is depending on the type of test and can not be generalised. For example, a url and a file size can be given to an according test function, which generates an upload request to a file server afterwards.

The test-server gets these test-atoms from the test automation system and is responsible to transmit them to the appropriate client. It also has to receive and store the result reports of all clients. Furthermore, test clients may have different status information, which can be necessary for decisions during the tests. These information has to be transmitted to the test-server and stored, similar to the connection parameters for this client.

4.3 Test-Automation

With the test-clients and the test-infrastructure it is possible to start primitive test on distributed clients from a central point and collect information about the success of these tests. Without an automation mechanism, a test engineer would be necessary, who generates the test-atoms.

Since it is very exhausting or impossible, to create a big number of such primitive tests in a short time, an test automation mechanism must generate these test-atoms based on a much more abstract test description. In the hypothesis of section 3 it is assumed, that the generation of a ”various amount of requests” is necessary for an efficient testing. Following this, it can be defined:

Definition 2: A test-scenario for a service i of an SPS is a function $f_i : T \rightarrow N$, which maps the times t 2 T to a value $n \in N$. Hereby n represents an absolut quantification of the service usage.

In other words, the input for the test-automation is a function, which tells, how much load or traffic should be generated to a service of the SPS at a point in time. It is important to announce, that the load or traffic, which should be generated to the service, must be quantifiable and controllable somehow. Otherwise a test with a defined ”amount of requests” is not possible.

Furthermore, it can be necessary for some kinds of tests, to restrict the set of possible clients, which can participate on this test. Since *f* is not able to describe this aspect, another function is necessary:

Definition 3: A test-restriction for a test-scenario f_i is a function $gi : C \rightarrow \{0, 1\}$, which maps the clients $c \in C$ to 0 *(don't use client in this scenario) or* 1 *(use client).*

Both functions fi and gi are inputs for the test-automation (shown in fig. 4) and has to be defined by the test engineer. They are necessary for each SPS service i which should be tested, e.g. file upload and file download.

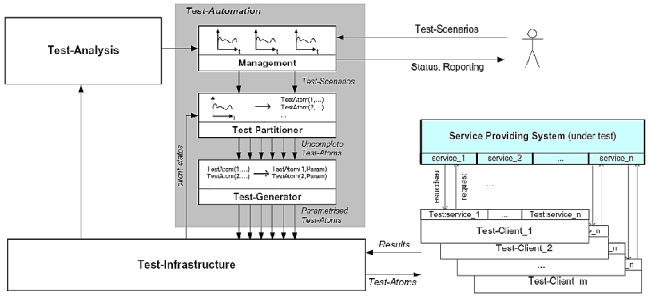

Fig. 4. Detailed overview of the test-automation component

The sets of functions $f = \{f_1, ..., f_n\}$ and $g = \{g_1, ..., g_n\}$ are inputs for a test partition algorithm, which has to decompose the abstract usage quantification value n_i to a set of test-atoms. Hereby it has to be ensured, that $g_i(j) = 1$ for all clients *j*, who are chosen to participate on the testscenario *i*. Beside this, the partitioner can use information about actual test-atoms, running on the clients. Furthermore, it can include information about technological capabilities of different clients, to decide which clients should be used for the scenario.

The test-atoms created by the test partitioner are not complete. They only include the definition of the test clients, who should be assigned with a test, and the function, which should be used. The parameters, necessary for the test, are appended by the test-generator.

The information about necessary parameters and possible values for these parameters depends on the used client and the test function, which is chosen to be started. These information has to be provided by the developer of the test function and can, for example, be stored in a database. If several values for a parameter can be valid for a test-atom, another function is needed, which helps the generator to decide. Details about formals of this function will be published later.

The generated test-atoms are now complete and can be sent to the according clients, using the test-infrastructure.

4.4 Control Loop

By definition, a test-scenario is a service usage quantification depending on time. So the required usage can change during time. Furthermore, the service requests of a test-atom will terminate after a period of time. This period is not necessarily constant or known.

Therefore, the test-automation is not a static process, where test-scenarios and restrictions are decomposed to test-atoms which are executed on the test clients. Rather, it is a dynamical process, where the execution state of the test-atoms has to be observed by the test partitioner. This can be seen in fig. 4 as the feedback of "client states" from the test-infrastructure. The partitioner has to react in an appropriate way, if the actual test situation violates rules of the test-scenario. Usually, it will generate new test-atoms if former test have finished. This is similar to the control loops, known from electrical engineering.

4.5 Test-Analysis

Tests serve to find weaknesses or errors or to determine special characteristics of/in a system. So, the introduced testautomation and infrastructure are only a means to an end - namely getting information about the results of the executed tests and to analyse or use these information.

Based on the test reports, which are sent to the test-server by the clients after the execution of a test-atom has terminated, a system can be analysed. These reports, including information about start and end times of the test and the state information of the client, are stored in a database. This way, a visualisation of the test history is possible with appropriate tools, so that the engineer can evaluate the tests.

Furthermore, it would be a continuative dimension of automatic testing, if the test results can be used to influence the automated test process. For example, if an analysis algorithm can identify - somehow - that there may are dependencies in the error rates of two services, the test automation can try to provoke this dependencies by influencing the corresponding test-scenarios. This way, another - more abstract - control loop is created in the test system. Details about the test-analysis component are shown in fig. 5.

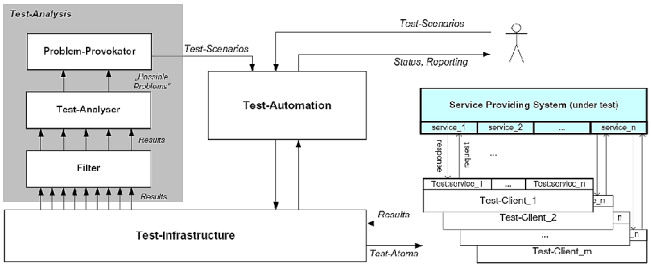

Fig. 5. Detailed overview of the test-analysis component

Before analysing, it is necessary to restrict the amount of test results by a filter. So unnecessary information or known but unimportant test problems can be removed. For example, if test-atoms of clients with a common operating system always fails and it is known as a problem of the OS, these can be filtered, because it is not a problem of the SPS.

The filtered values has to be analysed by an algorithm. It decides, if there are conspicuities, which has to be researched by further adapted tests. The results of this analysis are sent to the problem-provokater, which has to change the according test-scenarios or create new scenarios in an appropriate way.

It is obvious, that these feedback from the test results to the test-scenario is very important and a powerful tool for automated testing in general. So, filter and analysis techniques and formalism has to be an important part of future work.

5. Discussion

After presenting the test framework in detail, some advantages of disadvantages of the new system test approach are discussed in this section.

5.1 Application

Since the framework is designed to test a system by generating an amount of load or traffic to the provided services, the test of a wide range of systems is possible. The structure of these systems is hereby of no relevance. It can be a simple stand alone software as well as a complex distributed system composed of distributed software and hardware. The framework is defined in general and can be easily derived to a special test environment. Hereby, it only creates test cases for the SPS. The aim of the test can be defined by the test engineer. Typically, the aim is to find errors or weaknesses in the system. But it is also possible, to use the framework for performance or stress testing or to identify system parameters, which are necessary for runtime.

The system is mainly suitable for a application during field tests (see section 1). In opposite to the present practice, the test engineer gets the possibility, to execute own test-scenarios. So he is not constrained by the fluctuating and unpredictable behaviour of test users.

An interesting application would be the usage of the test system beyond field test - which is usually the real world runtime. For example it can be used for diagnostic of the running system. Therefore it is necessary, that the implementation of the test functions allows ans execution in background of normal user activity. Furthermore it has to be compliant with legal conditions.

The test process can also used for laboratory tests - but with limits. Especially the test-infrastructure component can help, to get access to the clients during the test. Hence, the clients can be mobile devices too. In opposite, the usage of the testautomation component for laboratory test is problematic, since much more specialised testcases are necessary in this phase of development.

5.2 Test Process

To use the test framework a test engineer only has to implement the test functions for the clients, including the mechanism for remote access. And he has to define the testscenarios and restrictions.

Thenceforward the tests can run automated without interferences of the engineer or the users. So a lot of testcases can be produced without binding human resources. Because the test framework has no knowledge about the internals of the tested system it is necessary that the system developers have a look at the test results. They have to decide if failed tests are originated in system errors, which has to be analysed, or

can be ignored, e.g. if they reason in a stress situation.

To get further benefit, reasonable analysis methods must be found, to influence the test-scenarios automatically. This would allow to provoke testcases for supposed errors or weaknesses automatically and so increases the test coverage.

5.3 Test Framework

Two important questions exists in conjunction with the test framework. The first is, how long the delay of the control loop is (see section 4). Obviously, this depends on the used partitioning and generation algorithms. Currently, a rudimentary Integer Linear Program (ILP, [12]) model exists for the partitioning. Since ILP solving is known as NP-hard
([7]), it has to be part of future work, to find a heuristic algorithm, so that the control loop delay can be kept small. A lot of work in partitioning problems is done in the research of Hardware/Software CoDesign ([14]).

The second question is, how to model the quantity of service requests, which are not been executed by the test functions of the clients. These unregulated requests can be generated by the test users, if they can use the clients concurrently. For example, if a user starts downloading a big file from a server, he generates service requests to the SPS, which are not intended by the test framework. Since the influence to the test-scenario can be significant, it is preferable, to detect this user behaviour and include it in the calculation of test-atoms during the phase of partitioning.

Furthermore, it is also possible to use parts of the tested SPS for communication between test-server and test clients. So the remote access of the test-server to a test client generates service requests too. The resulting amount of these requests has also to be included to the test-scenario partitioning.

6. Conclusion and Outlook

In this work, a new approach for abstract system testing of service providing systems (SPS) was introduced. This approach bases on the hypothesis that a controllable generation of a various amount of requests by a various number of clients is an efficient way for system testing of these SPS.

Furthermore a framework was defined, that allows the definition of test-scenarios, which are mappings of time to a quantified service usage, and test-restrictions. These inputs are decomposed to test-atoms automatically, which are transmitted to test clients by a test-infrastructure. The compliance with the scenarios during execution of atoms is controlled continual.

Based on the results of the test-atoms, conclusions can be determined. It is aim of the presented test process, that this determination is done by algorithms and can be used to influence the test-scenarios. Therefore, research about possible algorithms has to be done in the future. One possibility can be the usage of artificial intelligence, which is mentioned in [8].

Actually, the test-infrastructure is implemented using an TCP/IP based network for communication. Client test functions were written, which start requests to a server. Formal and technological backgrounds for test-automation are defined and can be implemented and published in next time.

7. References

[1] IEEE standard for software unit testing. **ANSI/IEEE Std 1008-1987**, 29 Dec 1986.

[2] W.M. Chan and Q.H. Tu. "RITA, a realization of remote isdn testing through packet networks." In INFOCOM '89. **Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies**. Technology: Emerging or Converging? IEEE, pages 1130–1131 vol.3, 23-27 Apr 1989.

[3] Edsger W. Dijkstra. "The humble programmer." **Commun. ACM**, 15(10):859–866, 1972.

[4] Rolf Drechsler. **Advanced formal verification**. Kluwer Academic, Boston, 2004.

[5] Elfriede Dustin, Jeff Rashka, and John Paul. **Software automatisch testen**. Springer, Berlin, 2001.

[6] Mark Fewster and Dorothy Graham. **Software Test Automation: Effective use of test execution tools**. Addison-Wesley Professional, Harlow, 1999.

[7] Michael R. Garey and David S. Johnson. **Computers and intractability**. Freeman, New York, 1979.

[8] Robert M. Hierons. "Artificial intelligence methods in software testing. edited by mark last, abraham kandel and horst bunke." published by world scientific publishing, singapore, **series in machine perception and artificial intelligence**, volume 56, 2004, isbn 981-238-854-0. Softw. Test., Verif. Reliab., 15(2):135–136, 2005.

[9] Edward Kit. **Software Testing in the Real World**. Addison-Wesley, Wokingham, 1995.

[10] Hagen Paul Pfeifer. **Scalable fault management of evolved radio access for 4G networks**. Master's thesis, Hochschule Furtwangen, August 2007.

[11] Winston W. Royce. "Managing the development of large software systems: Concepts and techniques." In **Technical Papers of Western Electronic Show and Convention (WesCon)**, 1970.

[12] Alexander Schrijver. **Theory of Linear and Integer Programming**. Wiley, John Wiley and Sons, 1998.

[13] W.R. Simpson and J.W. Sheppard. "System complexity and integrated diagnostics". **Design & Test of Computers, IEEE**, 8(3):16–30, Sep 1991.

[14] J¨urgen Teich and Christian Haubelt. **Digitale Hardware/ Software-Systeme: Synthese und Optimierung**. Springer-Verlag, Berlin Heidelberg, 2007.

[15] Yixin Zhao, Ying Liu, Xia Yin, and Jianping Wu. "A feasible distributed test system applied in routing protocol test. In **High Speed Networks and Multimedia Communications 5th IEEE International Conference** on, pages 172–176, 2002.