TESSY – MANAGEMENT OF SOFTWARE TESTS

Roman Pitschinetz and Joachim Wegener

Daimler-Benz AG, Research and Technology, Alt-Moabit 96 a
D-10559 Berlin, Germany

Abstract: A software testing tool is most effective, when it works within an organizational context. Computer-aided test activities framed into a testing life cycle management reduce time, effort and cost. The test system TESSY provides support for both automation of dynamic unit testing and appropriate management of the whole testing life cycle. Practical trials of the test system started in 1994. First promising results indicate that the test efficiency can be enhanced significantly by using TESSY. Future work will focus on extensions of TESSY with respect to integration testing and automatic generation of test cases and test data.

Keywords: Dynamic tests, Automatic testing, Software Tools, Computer-aided Testing, Management.

1. INTRODUCTION

The management of software tests includes several basic issues (Royer, 1993): knowledge of major testing methods to control quality, test planning, efficient test case design, test automation, and monitoring of the test progress. Dynamic testing is of major practical importance for the quality assurance of software systems. Typically, it consumes 30 % - 50 % of the overall software development effort and budget (see e.g. Marks, 1992). Investigations of various software development projects in several divisions of the Daimler-Benz Group showed that the costs for software testing mostly arose for unit testing, integration testing, and system testing. On average, 34 % of the total testing expenses is spent on unit testing, 28 % on integration testing and approximately 27 % on system testing. The remaining 11 % is spent on specific tests like the examination of software-hardware interfaces.

Furthermore, the investigations also revealed significant time-consuming activities in managing the software test process. Using individual software testing tools intermediate manual work still remains to be done, including organizing test objects, preparing tests, and inquiring the status of test activities and test work products. These test work products, such as test cases, test data, expected results and actual values, grow very fast and, therefore, have to be managed appropriately. Moreover, test rigs, environments for regression testing and batch processes are to be configured. Fully integrated software testing tools manage the software test process and reduce the time spent on tedious and error-prone manual work.

Hence, it is very important to provide an overall support for the whole testing life cycle. Significant savings can be achieved, and the product quality can be improved by tools automating both the test activities involved in software testing and the management of the software testing process.

This paper describes the managing of software testing processes by means of the computer-aided testing tool TESSY. The following chapter contains a description of the central test activities required for a thorough test. Chapter 3 gives a short overview of the architecture and functionality of TESSY. Afterwards, the management of software test processes is described as it is provided by TESSY. First results from practical trials are reported in chapter 5. After some concluding remarks the paper closes with a short outlook on future work.

2. TEST ACTIVITIES

A systematic test comprises the following main technical activities: test case determination, test data selection, expected results prediction, test case execution, monitoring, and test evaluation. This structure facilitates a systematic procedure and the definition of intermittent results. Figure 1 shows these test activities and the relationship between them.

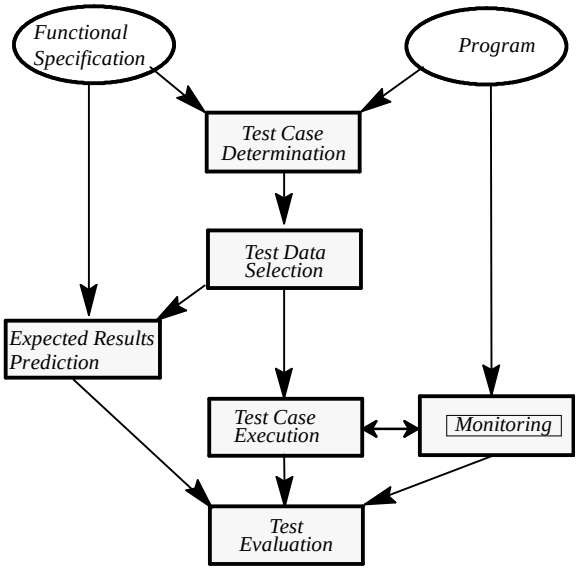

Figure 1: Test activities

In the course of test case determination, the test cases, with which the test object should be tested, are defined. A test case defines a certain input situation to be tested. It comprises a set of input data from the test object's input domain. Each element of the set should, for example, lead to the execution of the same program functionality, the same program branch or the same program state, depending on the test criteria applied. The test case determination is the most important activity for a thorough test, since it determines the kind and scope of the examination and thus the quality of the test. A test case abstracts from a concrete test datum and defines it, only in so far as it is required for the intended test. During test data selection the tester has to choose a concrete element from each test case with which the test should be executed.

Determining the anticipated results and program behavior for every selected test datum constitutes expected results prediction. If it is not possible to specify unequivocal output values or the expected behavior, acceptance criteria or other reference data have to be used for the predicition of expected results.

Subsequently, test case execution is performed. The test object is run with the selected test data. The output values and the program behavior are thus determined.

The behavior of the test object can be observed and recorded during test case execution by means of monitoring. A common method is to instrument the program code according to a white-box test criterion. For that purpose, the source code is extended by inserting statements at control-flow or data-flow-relevant points of the program, which count the number of executions of the corresponding program parts. Another kind of monitoring is performed by several capture-and-replay tools. They record the outputs produced by the test object on the screen and save them for regression testing.

In the course of test evaluation, actual and expected values as well as actual and expected program behavior are compared, and thus the test results are produced. Finally, the test is evaluated by comparing the test results achieved with the test objectives aspired to.

Due to the lack of software testing tools offering an overall and efficient support for all test activities, Daimler-Benz Research in Berlin developed the test system TESSY which provides general support for all test activities and aims at a considerable increase in unit testing efficiency for programs written in C. TESSY is designed to reduce the costs for software testing and to improve the reliability of systematic and thoroughly tested software products.

3. TEST SYSTEM TESSY

The test system TESSY was developed on VAXstations with the operating system VMS for unit testing of ANSI-C and VAX-C programs. Other C-compilers, especially for host-target cross-compiling, were added during the first half of 1995 corresponding to the needs of TESSY users. At the same time TESSY also became available on SUN/Solaris systems. Transfer to other UNIX systems is currently in progress.

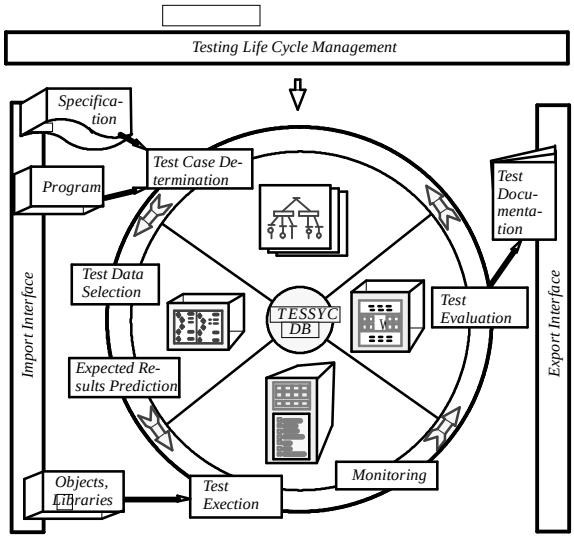

Figure 2: Architecture of TESSY

The most important strength of TESSY is that it provides support for the whole testing life cycle, offering a homogeneous, object-based, and context-sensitive graphical user interface which guides the tester

through all test phases (Wegener et al. 1994). Accordingly, the use of TESSY is simple and comfortable. As a result, the test system can be applied by testers without detailed knowledge of the C programming language. This promotes the distinction between software development and quality assurance as it is demanded in various standards. The user interface is based on OSF's Motif toolkit.

Figure 2 shows the architecture of TESSY with the main test activities in the center of the figure framed by the testing life cycle management and the accompanying import and export interfaces. Each test activity is supported by a window-based subtool exchanging data through the common database.

TESSY facilitates a combination of black-box and white-box tests according to the effective test strategy defined by (Grimm, 1989). The emphasis is laid on black-box testing since only test cases derived from the functional specification allow to examine appropriately, whether or not all specified requirements have been implemented into the test object. Test cases are determined using the classification-tree editor CTE, a graphical editor for the descriptive and systematic design of black-box test cases following the classification-tree method (Grochtmann et al. 1995). For the interactive input of test data and expected results, a browser tool navigates the tester through the test object's interface. Test case execution and test evaluation are totally automated. During the execution of black-box test cases the branch coverage is measured to estimate the test quality and its comprehensiveness. TESSY provides control panels with various options to control basic activities, such as test driver generation, coverage analysis and test rig configuration, and to specify the scope of the test documentation to be generated automatically. An additional white-box test is to be conducted, if the degree of branch coverage achieved is not sufficient to reach the test objectives. The test has to be improved subsequently by additional test cases derived from the program structure.

The testing life cycle management contains various tasks for managing the software test. They will be explained in the following sections.

4. MANAGING THE SOFTWARE TEST

An effective test requires support for both the automation of test activities and the management of the entire test process. As mentioned above, TESSY contains tools for each test activity. Furthermore, it provides powerful support for managing the software test, including test organization, test preparation, monitoring the status of test activities and test cases as well as recording the consumption of time for distinct activities. Moreover, test rigs, environments for regression testing and batch processes are configured automatically.

4.1. Test Organization

A testing tool is most effective when it works within an organizational context. Therefore, TESSY provides an appropriate test organization, which controls tool-invocation sequences and policies following the effective test strategy.

The technical framework of TESSY's test organization includes the import/export interface and the test database. Imports required for the test, e.g. the functional specification, program sources, object files, libraries and other information produced during the software development process, are managed. The database provides an export interface to easily transfer data to other tools such as project management tools or desktop publishing systems.

Furthermore, technical requisites for test case execution as, there is the implementation of stubs and test drivers, are organized. Another task is the configuration of make-files which generate test rigs by compiling and linking the test driver, the test object, stubs, object files and libraries. Host-target configurations are also produced automatically, and the remote process remains under control of the test organization.

Figure 3 shows the start-up window and several selection boxes to manage test projects. In TESSY a test is organized in projects. Each project contains at least one logical module and its export functions.

4.2. Test Preparation

In preparation of the test, the user has to supply some information for each module using the environment editor or the import interface. The essential data are the program sources for the logical module, the compiler to be used as well as compiler and linker options. In case all information is complete, TESSY investigates the whole export interface of the module automatically by analyzing all program sources stated. The export interface consists of all exported C-functions and their interfaces, including global variables, parameters, function return codes and the corresponding data types. The export functions ascertained represent the actual test objects.

Before the tester is able to start with the central test activities for each function, he has to complete the interface description of the export functions by entering certain characteristics of each interface component for which an automatic identification was impossible. He must specify the passing direction, that means he has to determine whether a component is only an input, an output, or input and output of the respective test object. Value parameters are always of the kind IN, the function return code is always of the kind OUT. Global variables and dynamic values, like pointers, can be of the kind IN, INOUT, or OUT. The completion of the interface description is carried out using TESSY's interface editor, a browser tool shown in Figure 4. The

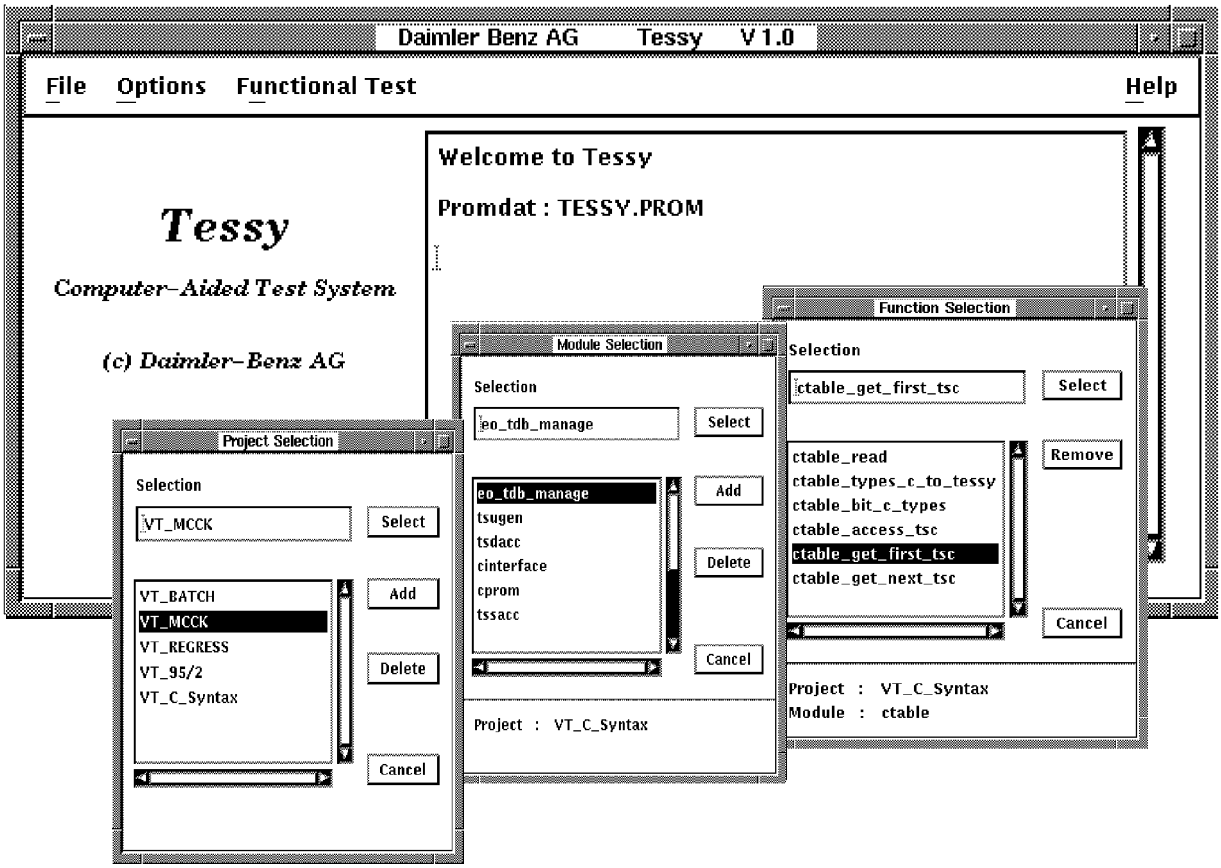

Figure 3: TESSY start-up window and selection boxes to manage test projects.

browser acts as a powerful navigation tool with support for collapse/expand and also partial views of the interface. It is possible to browse through complex structures down to the level of basic C data types. Figure 4 illustrates this for various parameters of different data types of a sample function.

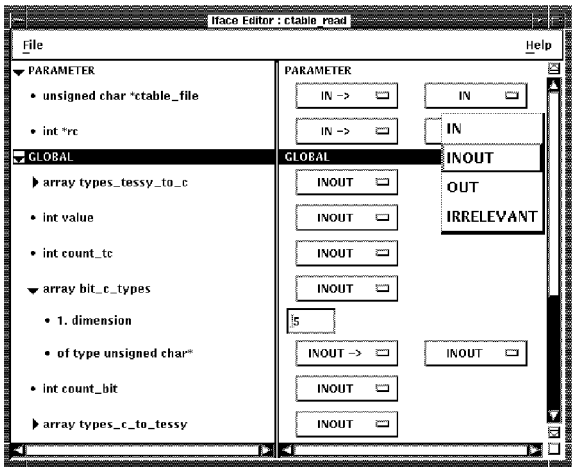

Figure 4: Editor for interface description

4.3. Test Measurements

To be able to control software testing, measurements are essential, especially for process maturity improvement (Humphrey, 1990). The impacts of maturity improvement aim at enlarged monitoring and tracking of the testing process, at better estimation of the software test development scheduling and at enhanced product quality. An appropriate mean for monitoring and tracking the software process is the status of test activities, test work products and test objectives. Based on time measurement, detailed efforts expended by each activity can be recorded as well as time required for project milestone. Thereby, productivity can be calculated. TESSY provides support for all these activities.

During the entire testing life cycle, TESSY exactly knows the status for each test object - whether or not the interface description is complete, test cases are defined, test data are selected, expected results are predicted, test cases are executed or test results are available. Test objectives, such as project specific failure rates or code coverage metrics, are determined and assessed automatically. The project manager offers the opportunity to monitor all test objects and modules of the project actually to be worked on.

Tessy-timer conducts time-measurement for each central test activity (Figure 5). It supports both automatic time-stamps and manual time measurement. In the first case, Tessy-timer records the time spent on each test activity automatically as long as the pertaining tools are used. In the second case, the user triggers the timer using buttons of Tessy-timer for time-outs or for

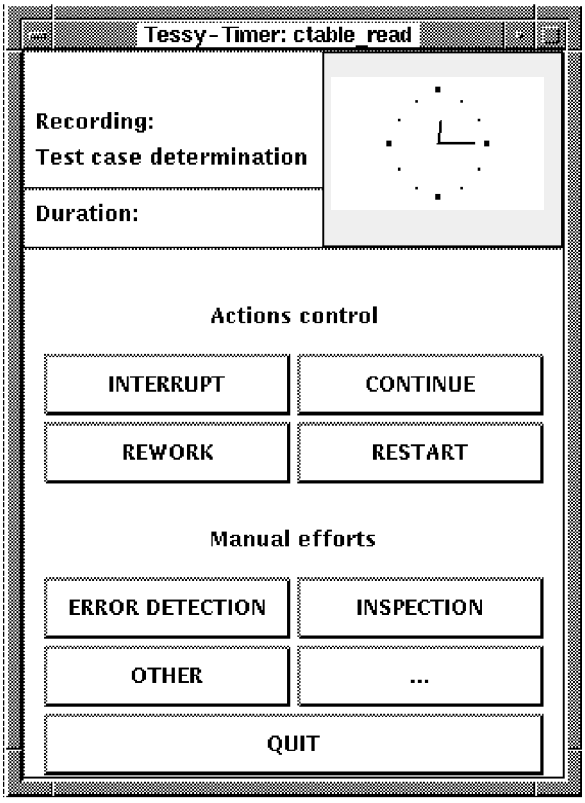

Figure 5: Tessy-timer

recording manual efforts. Furthermore, buttons can be added optionally if required.

The status of test objects and the counting times of associated activities are stored in the test system database. Based on the collected data detailed test management status tables can be generated, also including statistics. Measured time consumed by test activities is opposed to metrics of test work products, like the complexity of the test object's interface, lines of code, source statements, source branches, test cases, test data, expected results, imported functions, test results and errors. An examination of (Davey et al., 1995) showes that the best basis for an estimation of the expected testing effort is simply the number of source code statements.

However, it has to be taken into account that measuring of human productivity has an important psychological impact. Our experience shows that engineers do not mind quality criteria targets and the collection of data to achieve their goals as long as they participate in setting these goals and agree with them. Furthermore, the mere act of measuring human process changes them and thus enhances the product quality.

4.4. Batch Process

The use of batch processes enables large quantities of tests to be run in parallel and unattended, for example overnight or at weekends. Re-running all previous tests in one step can be performed at each time required by quality assurance or customer. A status protocol informs about successful and erroneous test runs. Furthermore, test documentation is generated for the whole project so that both quality assurance and customer can always inspect the current test version.

4.5. Regression Testing

In practice, tests are run an average of eight times - if done manually (Graham et al., 1995). Through the extensive management assistance provided by TESSY, comprising an integrated database for all test-relevant data, regression testing can be totally automated in most cases. After the tester has edited the required data for the first test execution, the data are available for further tests at any time. Normally, the test run simply has to be repeated with the new test object. Based on first results from practical trials, TESSY is to be expanded in 1995 by mechanisms which support regression testing also in cases where the test object interface has changed.

5. PRACTICAL EXPERIENCE AND RESULTING EXPANSIONS

In the fourth quarter of 1994, practical trials of the entire test system started in larger projects of Daimler-Benz divisions. The first application was just completed successfully. The test efficiency could be improved significantly. Savings of up to 70 % of the efforts normally required were estimated by the TESSY users due to the fact that TESSY totally automates several time-consuming activities. Nevertheless, some useful enlargements were proposed and TESSY was expanded to offer new functionality for batch processing and host-target testing. For example, several common target compilers are now supported. Furthermore, new import/export interfaces were added to supply TESSY with environment information available from other software development phases as well as with external test data and external reference data for test evaluation. This is very useful for large data, like audio and video data, available from the environment. Moreover, two additional interactive tools are currently under development to complement the support for regression testing. They can be applied to reuse test information like test cases, test data, and expected values in cases where the test object interface was changed or the classification-tree was enlarged by supplementary classifications and classes. It is also possible to use these tools to recycle existing test information for the examination of other test objects.

6. CONCLUSION AND FUTURE WORK

The test system TESSY provides powerful editors for test case determination, test data selection, and expected results prediction, as well as interactive tools

for managing the software test. They are specialized on the respective activities and conducive to a systematic test. The classification-tree editor CTE is of special importance, because it supports a thorough and well-structured test case determination corresponding to the classification-tree method. Test case execution, monitoring, test evaluation, and test documentation are executed automatically by TESSY (Table 1).

First practical trials of the test system were successful and gave several indications of further improvements. Next, TESSY has to be tried out in usual business. Assessments will be complemented by several measurements using Tessy-timer.

In the future, expansions of TESSY are planned to enable fully-automated unit testing. Owing to the strengths and growing relevance of formal methods in software engineering the use of formal specification techniques is planned. This will enable computer-aided generation of classification trees and test cases as well as automatic generation of test data from formal test case specifications. Test evaluation will be supported by executable specifications.

Other fields of future work will be the further support for integration and system testing as well as a test system version for Ada programs.

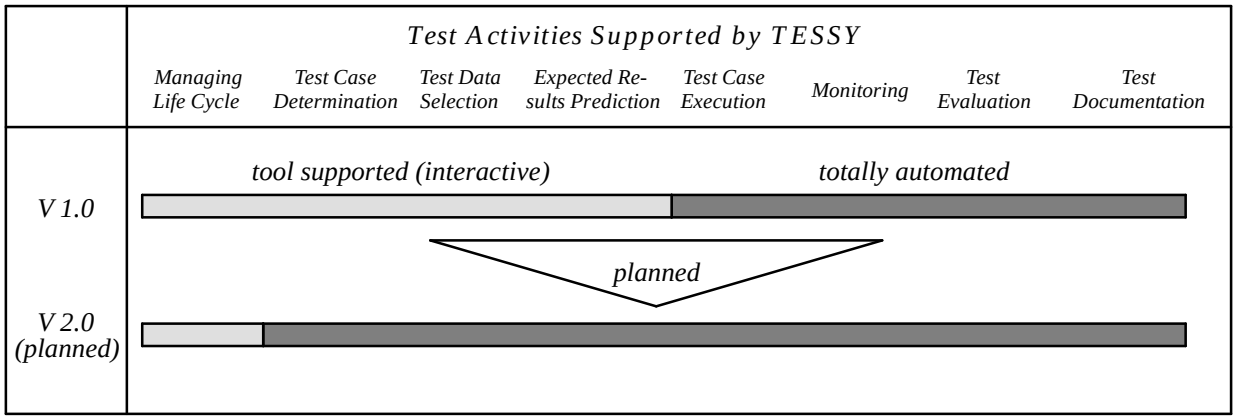

Table 1: Degree of automation provided by TESSY

REFERENCES

Davey, S., D. Huxford, J. Liddiard, M. Powley and A. Smith (1993). Metrics Collection in Code and Unit Test as Part of Continuous Quality Improvement. Proceedings of EuroSTAR '94 - 2nd European International Conference on Software Testing, Analysis, and Review, London, UK, pp. 48/1 - 48/27.

Graham, D.R., P. Herzlich and C. Morelli (1995). Cast Report - Computer-Aided Software Testing. Cambridge Market Intelligence Limited, London House, Parkgate Road, London, UK.

Grimm, K. (1989). An Effective Strategy and Automation Concepts for Testing of Safety-Related Software. In R. Genser et al. (Ed.): Safety of Computer Systems 1989 (SAFECOMP '89), Proceedings of the IFAC/IFIP Workshop, Vienna, Austria, Pergamon Press, UK, pp. 71 - 79.

Grochtmann, M., J. Wegener and K. Grimm (1995). Test Case Design Using Classification Trees and the Classification-Tree Editor CTE. Proceedings of 8th International Software Quality Week, San Francisco, California, USA, pp. 4-A-4/1-11.

Humphrey, W. S. (1990). Managing the Software Process. Addison-Wesley Publishing Company, Inc., New York.

Marks, D. M. (1992). Testing Very Big Systems. McGraw-Hill, Inc., New York.

Royer, C. R. (1993). Software Testing Management. PTR Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Wegener, J., R. Pitschinetz, K. Grimm and Grochtmann, M. (1994). Tessy – Yet Another Computer-Aided Software Testing Tool? Proceedings of EuroSTAR '94 - 2nd European International Conference on Software Testing, Analysis, and Review, Brussels, Belgium, pp. 36/1 - 36/13.