

How to Test Software

Tanja Toroi

Report

Department of Computer
Science and Applied

Mathematics

University of Kuopio

March 2002

Contents

1	INTRODUCTION	3
2	TESTING AND INSPECTION PROCESS	3
3	TEST MANAGEMENT.....	7
4	TEST CASE CREATION.....	7
4.1	WHITE BOX TESTING	8
4.1.1	Control-flow testing	8
4.1.2	Data-flow testing.....	11
4.2	BLACK BOX TESTING	11
4.2.1	Equivalence partitioning.....	12
4.2.2	Boundary value analysis	14
4.3	TEST CASE DEFINITION	15
5	SOFTWARE TESTING TOOLS.....	15
6	TESTING COMPONENT SYSTEMS	16
7	FURTHER RESEARCH PROBLEMS.....	17
	LITERATURE	18
	APPENDICES	
A	Testing business component systems	
B	Sample test plan	
C	How to derive test cases from user requirements	
D	Rational Test Tools	
E	Software testing tools	
F	Rational Unified Process	

1 Introduction

This report reviews testing results of the PlugIT/TEHO project's first phase. Testing and inspection processes are described in Chapter 2. Test management is shortly described in Chapter 3. Test case creation by white box and black box testing methods is described in Chapter 4. In Chapter 5 software testing tools are handled. Testing component systems is described in Chapter 6 and finally in Chapter 7 some further research problems are mentioned.

Comments and feedback can be sent to Tanja.Toroi@cs.uku.fi. Thank you!

2 Testing and inspection process

In this Chapter there is a preliminary description of software engineering process. The description will be specified in the PlugIT-project phases 2 and 3. The following themes will be taken into consideration:

- Forward and backward traceability.
- Component provisioning and assembly.
- Component repository.
- Testing process, which differs according to the role of component tester. A tester can be:
 - provider who uses black and white box testing techniques,
 - integrator who needs black box testing techniques and performs interface testing, and
 - customer who uses black box techniques for acceptance testing.

Testing and inspection processes are organized in levels according to the V-model (see Figure 1.).

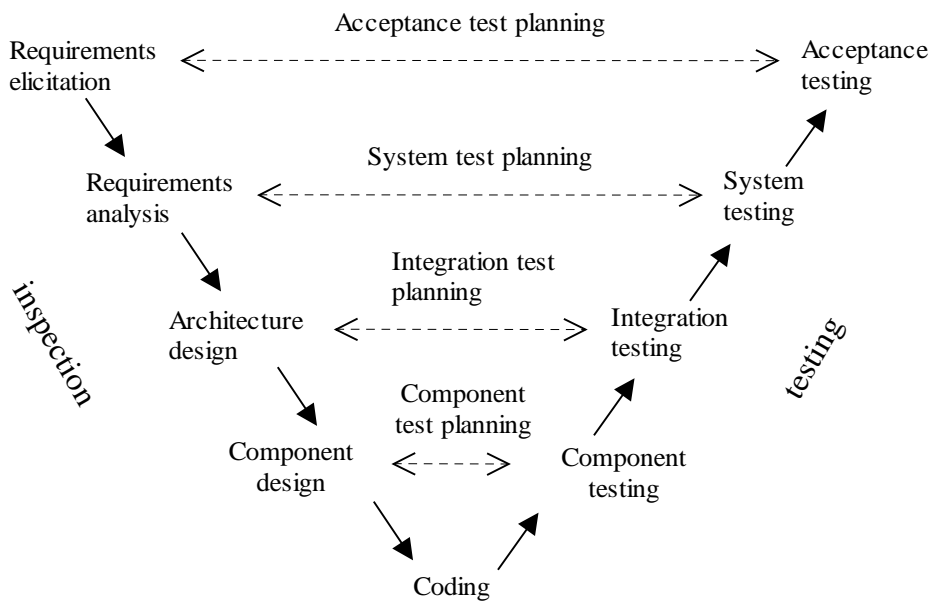


Figure 1. V-model of software testing

The V-model integrates testing and construction by showing, how the testing levels (on the right) verify the constructive phases (on the left). In every construction phase corresponding test plan is made. Requirement specification document and design documents are inspected carefully before the coding phase begins. In inspection the checklists are used.

Requirements elicitation involves listening users' needs or requirements, asking questions that assist the users in clarifying their goals and constraints, and finally recording the users' viewpoint of the system requirements. Requirements give us guidelines for the construction phases and give the acceptance criteria for the software or the software component. Acceptance criteria are used in acceptance testing. In the requirement elicitation phase the acceptance test plan is made. Acceptance test plan helps testing when users test the system and check whether their requirements are fulfilled.

Requirement analysis is the stage where the software engineer thinks about the information obtained during the first stage. He transforms the users' view of the system into an organized representation of the system and writes clear statements of what the system is expected to provide for the users. The end product is called a requirement specification document. In the requirement analysis phase the system test plan is made. System test plan is used in the system testing phase. During the requirement specification use cases and conceptual models (for example class diagram) are designed.

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them [BaC98]. The design of software architecture can be divided into

- functionality-based architecture design, which decomposes the architecture into needed components and the relationships between them,
- evaluation of quality attributes for architecture , and
- transformation of software architecture in order to improve quality [Bos00].

Component design means that the structure of each individual component is designed. A component structure is presented in Figure 2. In component design phase all services the component offers are defined accurately. Interface includes each operation the component has, parameters for each operation and type, which tells the direction of the operation (in or out). The component can not be isolated, but it must collaborate with other components. So, dependencies from the other components must be designed, too. Component's functional logic is often object-oriented. Thus we must design classes and relationships between them. Functional logic is separated from interface and dependency implementation. Here we need proxies. Component execution environment is defined, too. The component test plan is made in this phase.

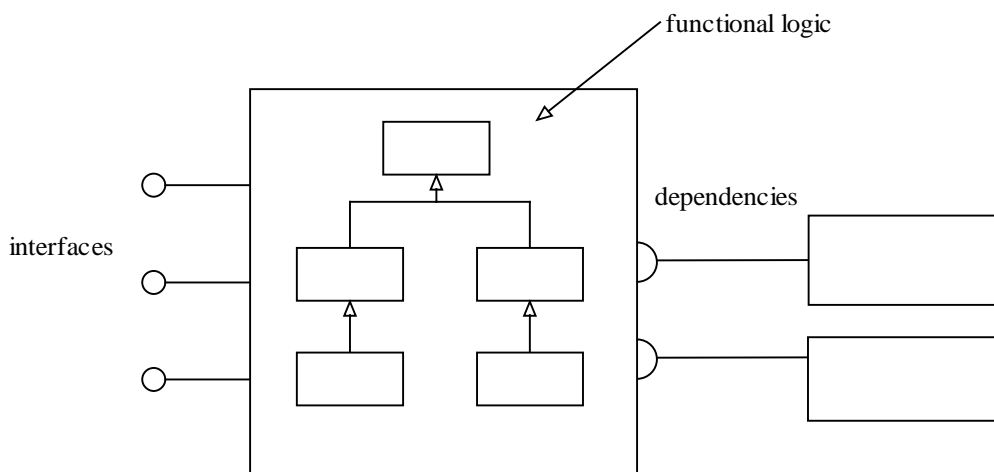


Figure 2. Component structure

In the coding phase software components are implemented according to the component design document. They can be implemented with traditional, object-oriented or component-based techniques.

In component testing phase each individual component is tested on the source code level. The term component refers to a lowest level software module. Each programmer uses white box testing methods and tests his own components. We need test stubs and drivers to simulate components, which are not yet ready. Test stub is a component, which accepts all the input parameters and passes back suitable output data that lets the testing process continue. The stub is called by the component under test. Test driver represents a component that calls the components under test and makes them executable. Component testing is based on component design documents.

In integration testing phase the previously tested software components are integrated and tested incrementally. Integration is done either top-down or bottom-up. In top-down integration the top-level component is tested by itself. Then all the components called by the components already tested are integrated and tested as a larger subsystem. The approach is reapplied until all the components have been integrated. In bottom-up integration each component at the lowest level of the system hierarchy is first tested individually. Then are tested those components, which call the previously tested components. This principle is followed repeatedly until all the components have been tested. The bottom-up approach is preferred in object-oriented development and in systems with a large number of stand-alone components. The focus of integration testing is on the cooperation of components and their interfaces. Test stubs and drivers are needed to simulate not-tested components. When integrating distributed components, the business components are formed. When integrating business components, business component systems are formed (see Chapter 6). Integration testing is based on architecture design documents.

In system testing phase the tested business components are integrated and the system is tested as a whole business component system. The goal of the system testing is to demonstrate in what degree a system does not meet its requirements and objectives. System testing is based on requirements document. Requirements must be specific enough so that they are testable. System testers can be both software engineers and actual users. System testing can have several different forms, such as performance testing, volume testing, load/stress testing, security testing and configuration testing.

In acceptance testing phase customers and users check that the system fulfills their actual needs and requirements. Usability is taken into consideration in this phase. Acceptance testing is based on requirement specification. Acceptance testing can take forms of beta or alpha testing. Beta testing is performed by the real users and in a real environment. Alpha testing is performed by the real users at the developing company. In integration, system and acceptance testing phases the black box testing methods are used.

The whole testing process depends on those who are testing the components. The process is different if the tester is provider, integrator or customer of the component.

3 Test management

Systematic testing has to be planned and executed according to the plan and it has to be documented and reported. There is an example of the test plan document in Appendix B.

Test case specification defines test cases, their inputs, expected outputs and execution conditions. Test cases can be created using white box or black box techniques (see Chapter 4).

Test report (bug report) is generated after each test. It describes defects found and what is the impact of the defect (critical, major, minor, cosmetic). The most critical defects are always fixed. They stop the user from using the system further. Major defects stop the user from proceeding in the normal way but a work-around exists. They can also be in many components and the engineer needs to correct everything. Minor defects cause inconvenience but they do not stop the user from proceeding. Cosmetic defects can be left unfixed if there is not enough time and resources. A cosmetic defect could be for example an aesthetic issue. Test report describes also the inputs, expected outputs and actual outputs, test environment, test procedure steps, testers, repeatability (whether repeated; whether occurring always, occasionally or just once) and other additional information.

4 Test case creation

We started to study test case creation in spring 2002. In this Chapter basic testing methods, white box testing and black box testing are described with examples. It should be remarked that the

testing tools for the methods presented here are available on the market. An example of granular test cases is shortly described in Chapter 4.3. The complete example can be found in Appendix C.

4.1 White box testing

Test cases can be created using white box or black box testing techniques. If the source code is available the white box testing techniques are used. White box techniques ensure that the internal logic of the system have been adequately tested. In white box testing test cases are derived examining carefully the source code of the component. White box testing can find errors that are deeply hidden in the details of source code. White box testing is used in component testing phase (see Figure 1.).

4.1.1 Control-flow testing

When testing the component in source code level the control-flow and the data-flow of the component is tested. Control-flow testing means that different paths according to the control-flow of the component are tested. A standard representation for the control-flow of a component is a flow graph, which abstracts the execution of the component into a graph-like structure. The nodes of a flow graph stand for the statements of the component and the edges stand for the control transfer between the statements. Statements where the control diverges such as conditional statements or loops are the most important from the control-flow testing point of view. Below there is a method called liability. A flow graph for the method is in Figure 3.

```
public class patientPayment {
    int payment;
    public void liability (int age, String sex) {
        if (sex == "male") {
            if ((age > 17) && (age < 36))
                payment = 100;
            else if ((age > 35) && (age < 51))
                payment = 120;
            else if (age > 50)
                payment = 140;
        }
        else if (sex == "female") {
            if ((age > 17) && (age < 31))
                payment = 80;
            else if ((age > 30) && (age < 51))
                payment = 110;
            else if (age > 50)
                payment = 140;
        }
    }
}
```

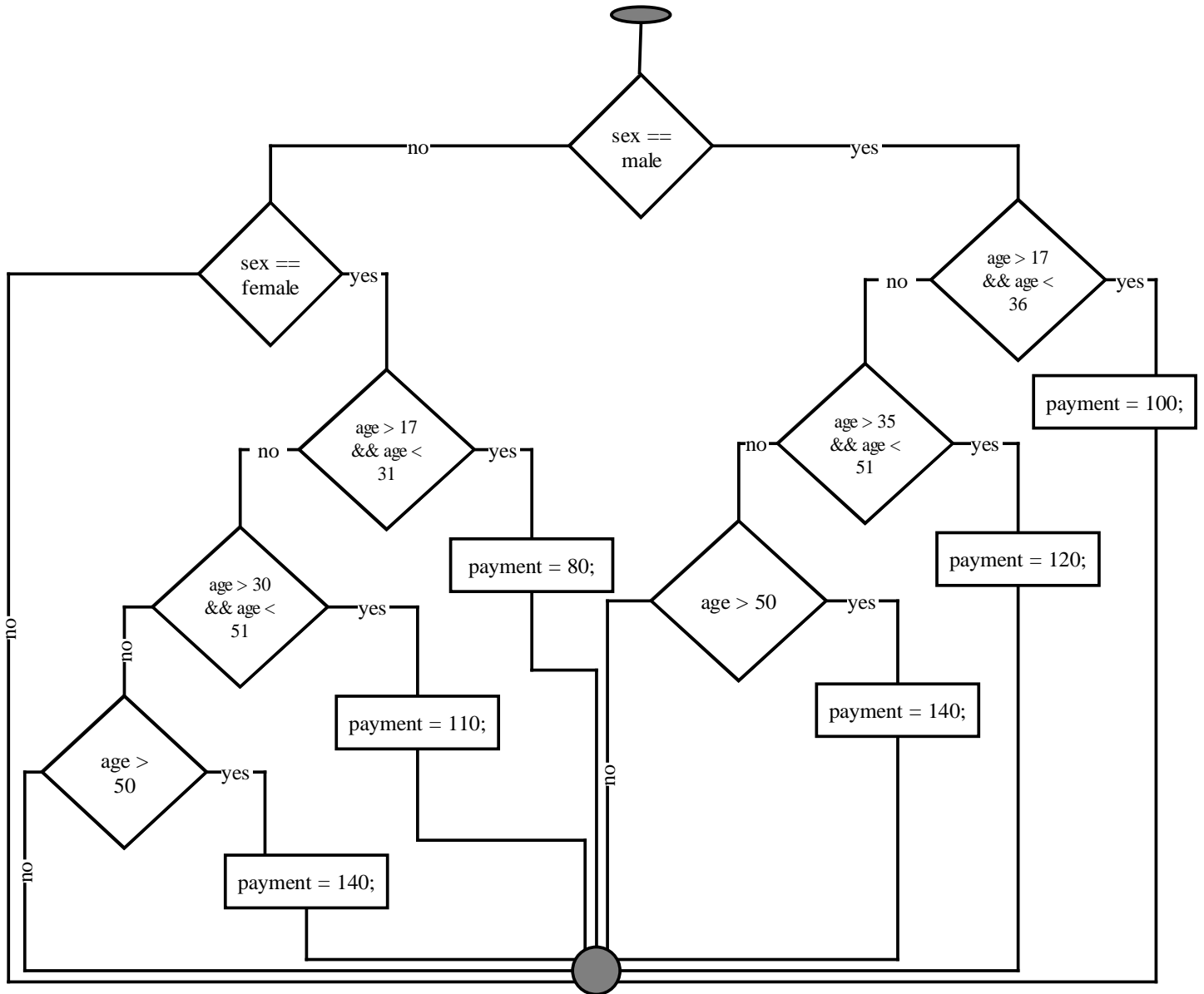



Figure 3. Control-flow graph for liability method

The adequacy of control-flow testing is measured in terms of coverage. The coverage indicates how extensively the system is executed with a given set of test cases. There are five basic forms of coverage: statement coverage, branch coverage, condition coverage, multiple condition coverage and path coverage.

- In statement coverage each statement is executed at least once. This is the weakest criterion and does not normally ensure a faultless code. On the other hand 100% statement coverage is usually too expensive and hard to achieve, especially if the source code includes "dead code" (= a code, which can never be reached).

- In branch coverage each statement is executed at least once and each decision takes on all possible outcomes at least once. Branch coverage criterion is stronger than that of statement coverage because if all the edges in a flow graph are traversed then all the nodes are traversed as well.
- In condition coverage each statement is executed at least once and each condition in a decision takes on all possible outcomes at least once. Complete condition coverage does not necessarily imply complete branch coverage so they do not compensate each other.
- In multiple condition coverage each statement is executed at least once and all possible combinations of condition outcomes in each decision occur at least once. This is the strongest criterion and forces to test the component with more test cases and in more detail than the other criteria.
- In path coverage every possible execution path is traversed. Exhaustive path coverage is generally impractical and impossible because loops increase the amount of execution paths.

The essential differences between statement coverage, branch coverage, condition coverage and multiple condition coverage can be summarized by the following example (see Figure 4.).

```
if ((sex == "male") && (age > 50)) payment = 140;
```

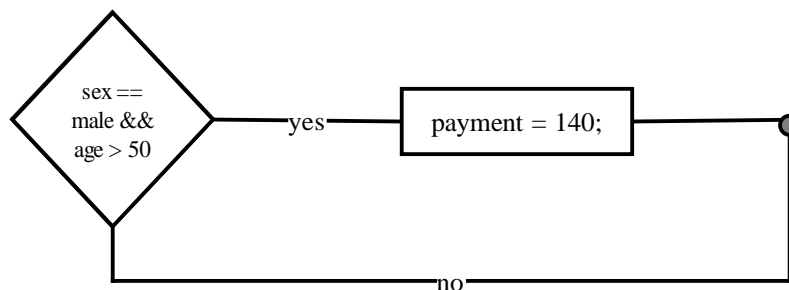


Figure 4. Example of differences of coverage criteria

There are a minimum number of test cases for each coverage criterion.

- Statement coverage needs only a single test case:
 - (sex = male, age = 51).
- Branch coverage can be reached by two test cases:
 - (sex = male, age = 51) for the yes-branch and
 - (sex = female, age = 51) for the no-branch.

- Condition coverage needs also two test cases:
 - (sex = male, age = 51) for the atomic combination (true, true) over the control predicate (sex = male) && (age > 50) and
 - (sex = female, age = 11) for the atomic combination (false, false).
- Multiple condition coverage can not be reached with fewer than four test cases:
 - (sex = male, age = 51) for the atomic combination (true, true),
 - (sex = male, age = 12) for the combination (true, false),
 - (sex = female, age = 52) for the combination (false, true), and
 - (sex = female, age = 22) for the combination (false, false).

Multiple condition coverage is the most effective criterion of these because it forces one to test the component with more test cases than the other criteria.

It has to be remarked that different testing tools can and should be used to check coverage criteria and to help testing (see Appendix D (Rational PureCoverage) and Appendix E (Test Evaluation Tools)).

4.1.2 Data-flow testing

Data-flow testing methods explore the events related to the status of variables during component's execution. The central events related to the variables are the assignment of value (for example $\underline{x} := 1$) and uses of value (for example `if $\underline{x} < 0$ then`).

4.2 *Black box testing*

Black box testing techniques are used if the source code is not available. Black box techniques ensure that the system fulfills users' requirements. Black box testing finds errors that are related to the requirements. Both white box and black box methods should be used so that testing is as covering as possible. In black box testing test cases are derived from requirement specification, use cases or contracts. A contract defines interfaces the component has, dependencies from other components and component execution environment. Use cases are available for business components and business component systems (see Chapter 6). Contracts are normally used for testing distributed components. Black box testing is used in integration, system and acceptance testing phases.

4.2.1 Equivalence partitioning

Commonly used black box testing methods are equivalence partitioning and boundary value analysis. In most cases the system can not be exhaustively tested, so the input space must be somehow partitioned. Equivalence partitioning is a test case selection technique in which the tester examines the entire input space defined for the system under test and tries to find sets of input that are processed "identically". The identical behavior means that test inputs in one equivalence class traverse the same path through the system. Equivalence classes are defined based on requirement specification document. The hypothesis is:

- If the system works correctly with one test input in an equivalence class the system works correctly with every input in that equivalence class.
- Conversely, if a test input in an equivalence class detects an error, all other test inputs in the equivalence class will find the same error.

However, equivalence partitioning is always based on tester's intuition and thus it may be imperfect.

Equivalence classes can be designed according to the following guidelines:

1. If the input specifies a range of values, one valid (within the range) and two invalid (one outside each end of the range) equivalence classes are defined. Example: If the input requires a month in range of 1-12, define the following equivalence classes: $\{1 \leq \text{month} \leq 12\}$, $\{\text{month} < 1\}$ and $\{\text{month} > 12\}$.
2. If the input specifies a specific value within a range, one valid and two invalid equivalence classes are defined. Example: If the input condition states that a value of integer x must be 1, define the following equivalence classes: $\{x = 1\}$, $\{x < 1\}$ and $\{x > 1\}$.
3. If the input specifies a set of valid values, one valid (within the set) and one invalid (outside the set) equivalence class are defined. Example: If the input requires one of the weekdays Monday, Tuesday, Wednesday, Thursday or Friday to be an input value, define the following equivalence classes: $\{\text{Monday, Tuesday, Wednesday, Thursday, Friday}\}$ and $\{\text{Saturday, Sunday}\}$.
4. If there is reason to believe that the system handles each valid input value differently, then define one valid equivalence class per valid input.
5. If there is reason to believe that elements in an equivalence class are not handled identically, subdivide the equivalence class into smaller equivalence classes.

6. One or several equivalence classes are always defined for the illegal values. Illegal value is incompatible with the type of the input parameter (for example, if the valid input value is integer, illegal value is real, char or string).

Usually, the input does not consist of a single value only but from several consecutive values. In that case the equivalence classes for the whole input can be designed as combination of the equivalence classes for the elementary domains. For example, if the component expects as input two integers x and y that both shall be within the range 1...10000, the following equivalence partitioning can be defined:

- for x: one valid, two invalid (too small, too large) and one illegal (for example, real) equivalence class
- for y: one valid, two invalid and one illegal equivalence class

for (x,y) the combination of the above classes:

- (x valid, y valid) accepted values
- (x valid , y too small) } error message
- (x valid, y too large) }
- (x valid, y real) }
- (x too small, y valid) }
- ...

There are totally $4 * 4 = 16$ equivalence classes.

This leads to an explosion in the number of equivalence classes. There is only one valid equivalence class and the most of the obtained classes consist of input values that the component should not accept. In practice it is not sensible to test the component with all the invalid and illegal combinations of input values but to select classes so that each invalid and illegal class have been covered at least once.

When equivalence classes have been identified test cases have to be selected. It is assumed that all the values in an equivalence class are identical from the testing point of view. We can select any test case in an equivalence class so that at least one test case is selected from every class to the execution. When using equivalence partitioning the testing is on one hand effective and covers customers' requirements and on the other hand it is not too arduous and complex because we can reduce test cases effectively and still test the component adequately. Redundancy is also as mini-

mal as possible. Equivalence partitioning can be improved by boundary value analysis (see Chapter 4.2.2).

There is an example of how to derive test cases from user requirements in Appendix C. First there is a Patient management application, which consists of one dialog. Test cases are derived using equivalence partitioning. Second there is an application with many dialogs. Now test cases are derived from use case diagram and activity diagram. Finally there is an example of testing action flow of many people. In this case test cases are derived from use cases.

4.2.2 Boundary value analysis

Boundary value analysis is a variant and refinement of equivalence partitioning. Several studies have shown that programmers make errors especially when coding loops. Boolean expressions, such as $<$, $>$, $=$, \neq , \geq and \leq are often erroneously coded and the loop is traversed one too much or one too few times. That is why we should select test cases close to the boundaries, not from the center of the equivalence class. This is illustrated in Figure 5. The guidelines for the boundary value analysis are:

1. If an input specifies a range of valid values, write test cases for the ends of the range and invalid input test cases for conditions just beyond the ends. Example: If the input requires a real number in the range 1.0...9.0, then write test cases for 1.0, 9.0, 0.9999 and 9.0001.
2. If an input specifies a number of valid values, write test cases for the minimum and maximum number of values and one beneath and beyond these values. Example: If the input requires at least one forename for a patient, but no more than 3 forenames, write test cases for 0, 1, 3 and 4 names.

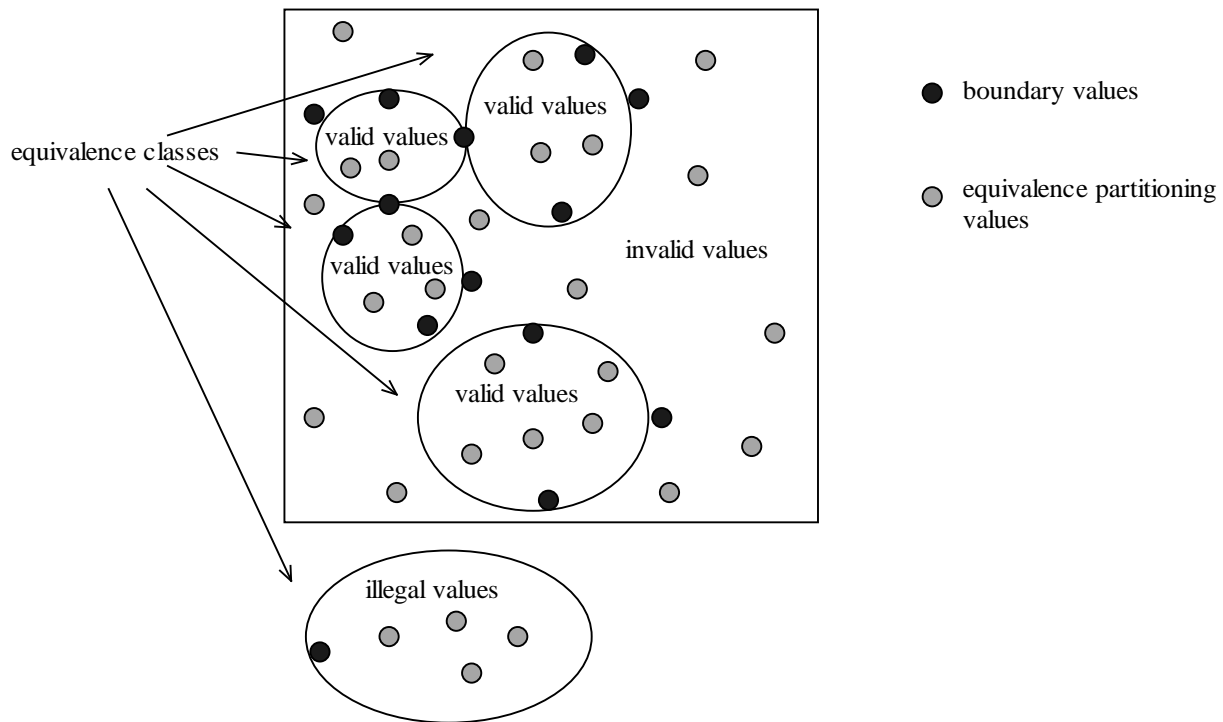


Figure 5. Test cases by boundary value analysis

4.3 Test case definition

A test case is defined in Rational Unified Process as a set of test inputs, execution conditions, and expected results developed for particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. Test cases are derived from users' requirements, which are described with use cases or contracts. Use cases are available at the system and component level. Contracts are available at the lowest component level. We insert the granularity aspect in the definitions of test cases:

- Test case at the system level is an action flow between system and users.
- Test case at the component level is a sequence of operations between internal components of the component.
- Test case at the lowest level is a method sequence between object classes.

There is an example of granular test cases in Appendix C.

5 Software testing tools

Testing process should be assisted with specialized testing tools. Testing tools can be grouped into the V-model (see Figure 6.). Although the tools are used manual work is still needed. Testing pro-

cess can not be totally automated yet. We have evaluated Rational Test Studio tools, which includes Rational Administrator, Rational Test Manager, Rational Robot, Rational PureCoverage, Rational Purify, Rational Quantify and Rational TestFactory (see Rational Test Tools in Appendix D). Also other software testing tools have been investigated and they have been grouped into test design tools, graphical user interface test drivers, load and performance testing tools, test management tools, test implementation tools, test evaluation tools and static analysis tools (see Software testing tools in Appendix E).

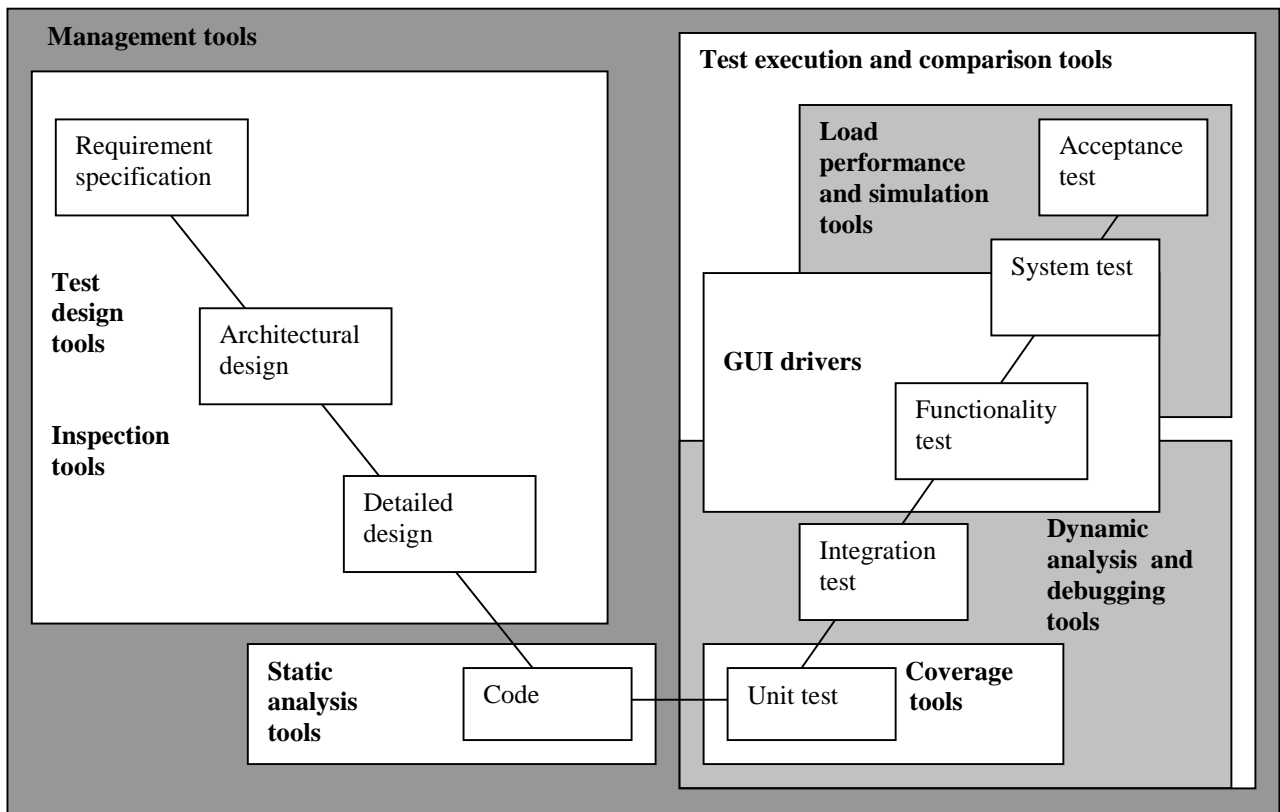


Figure 6. Division of the tools in V-model

6 Testing component systems

Traditional techniques such as data flow testing, mutation testing, and control flow testing are not very well applicable when testing component-based systems. Testing distributed component-based systems requires that heterogeneity, source code availability, maintainability and complexity need to be considered. Testing business component systems is presented in [TEM02]. An effective and practical method for testing business component systems step by step is presented. Components are from different granularity levels defined in [HeS00]. The granularity hierarchy means that a business component system is a composition of business components, which in turn are composi-

tions of lower level components. The advantages of this approach are the possibility to master development and deployment complexity, to decrease time to market, and to increase scalability of software systems. Also the great number of dependencies which occur in object-oriented approach can be mastered better, because majority of the dependencies between classes remain inside one component where the number of classes is much less than in the total system or subsystem. The abstraction levels decrease the work needed in testing, because the testing work can be divided into sufficiently small concerns and previously tested components can be considered as black boxes, whose test results are available. Furthermore, errors can be easily detected, because not so many components are considered at one time. Testing business component systems can be read in more detail in Appendix A.

7 Further research problems

- Test case selection (harava-ongelma)
- Regression testing, component regression testing and test case selection in regression testing
- Object-oriented testing
 - antidecomposition = Testing a component in the context of an enclosing system may be adequate with respect to that enclosing system, but not necessarily adequate for other uses of the component.
 - anticomposition = Testing each individual component in isolation does not necessarily suffice to adequately test the entire system. Interaction can not be tested in isolation.
 - dependencies in object-oriented programs
 - object state testing
 - state can change to the appropriate new state
 - state can leave as it is
 - state can change to an undefined state (error)
 - state can change to a defined, but inappropriate state (error)
 - state can leave as it is when it is supposed to change (error)
 - object-oriented regression testing
- Debugging methods
- Testing distributed components
- Compatibility testing
- Component systems integration testing

- COTS (component-off-the-shelf)

Literature

- [BaC98] Bass L., Clements P., Kazman R.: Software Architecture in Practice, Addison-Wesley 1998.
- [Bos00] Bosch Jan: Design and use of Software Architectures Adopting and evolving a product-line approach, Addison-Wesley 2000.
- [Bel00] Bell, D.: Software engineering, A programming approach. Addison-Wesley, 2000.
- [DRP01] Dustin, E., Rashka, J., Paul, J.: Automated software testing, Introduction, Management and Performance. Addison-Wesley, 2001.
- [HaM01] Haikala, I., Märijärvi, J.: Ohjelmistotuotanto. Suomen Atk-kustannus, 2001.
- [HeS00] Herzum P., Sims, O.: Business Component Factory. Wiley Computer Publishing, New York, 2000.
- [Jal99] Jalote, P.: CMM in practice. Addison-Wesley, 1999.
- [Kit95] Kit, E.: Software testing in the real world. Addison-Wesley, 1995.
- [Mye79] Myers, G.: The art of software testing. John Wiley & Sons, 1979.
- [Paa00] Paakki, J.: Software testing. Lecture notes, University of Helsinki, 2000.
- [PeK90] Perry, D., Kaiser, G.: Adequate testing and object-oriented programming. J. Object-Oriented Programming, Jan./Feb. 1990, 13-19.
- [TEM02] Toroi, T., Eerola, A., Mykkänen, J.: Testing business component systems. 2002, (submitted).

Testing business component systems

Tanja Toroi

University of Kuopio

Department of computer science and applied mathematics

P.O.B 1627, FIN-70211 Kuopio

+358-17-163767

tanja.toroi@cs.uku.fi

Anne Eerola

University of Kuopio

Department of computer science and applied mathematics

P.O.B 1627, FIN-70211 Kuopio

+358-17-162569

anne.eerola@cs.uku.fi

Juha Mykkänen

University of Kuopio

Computing Centre

HIS Research & Development Unit

P.O.B 1627, FIN-70211 Kuopio

+358-17-162824

juha.mykkanen@uku.fi

ABSTRACT

Nowadays it is demanded that software system fulfils more quality requirements, especially in the health care and other safety critical systems. In this paper we present an effective and practical method for testing business component systems step by step. We utilize components of different granularity levels. The advantages of component-based systems are the possibility to master development and deployment complexity, to decrease time to market, and to support scalability of software systems. Also the great number of dependencies which occur in object-oriented approach can be mastered better, because majority of the dependencies between classes remain inside one component where the number of classes is much less than in the total system or subsystem. The abstraction levels decrease the work needed in testing, because the testing work can be divided into sufficiently small concerns and the previously tested components can be considered as black boxes, whose test results are available. Furthermore, errors can be easily detected, because not so many components are considered at one time.

In our method, components of different granularities are tested level by level. The idea of the method is that at each level white box testing and black box testing occur alternately. We define test cases based on component granularities at distributed component, business component and business component system level. Test cases are derived from use cases or contracts. We use a dependency graph, which shows dependencies between the same granularity components. The dependency graph is used to assure that the whole functionality of the component has been covered by test cases.

Categories and Subject Descriptors

D.2.5 [Software Engineering]: Testing and Debugging – *Testing tools*.

D.2.9 [Software Engineering]: Management – *Software quality assurance*.

General Terms

Design, Reliability, Theory, Verification.

Keywords

Testing, distributed component, business component, business component system, distribution tiers, layers, interfaces, dependency

1. INTRODUCTION

Component software technologies have been exploited in software industry more and more. The crucial objectives in this approach are decreased time to market, efficiency in software production, quality and reusability of software and its components. Distribution of software and its production is the fact today. Software providers do not necessarily want to implement all the properties of the system by themselves but they want to specialize in their strategic competitive edge and buy other properties as ready-made COTS (commercial-off-the-shelf) components. In situations like this testing and documentation are even more important than in conventional software projects. Customers of software require quality such as correctness and reliability in addition to functional properties of the software system. This is especially true in health care information systems and in other safety critical systems.

There are several definitions of the concept of testing [19]. In this research, testing means that the software is executed in order to find errors. Thus debugging process is not included in testing, although it is a quite near to it. In order to develop quality business components, system testing must verify that the software product operates as documented, interfaces correctly with other systems, performs as required, and satisfies the user's needs [26].

Although component technologies have been widely introduced in research and industry only a few investigations have been dedicated to testing of component based systems. Even less have integration testing and interoperability testing been considered in research papers. Unlike traditional software developers, component developers are in most cases independent software vendors who may not know all the future uses of their components [26]. Only the known uses can be tested beforehand. In order to increase the reusability we must carefully define the provided interfaces and constraints, i.e. runtime environment and dependencies, of the components.

Traditional techniques such as data flow testing, mutation testing, and control flow testing are not very well applicable when testing component systems. Testing distributed component-based systems requires that heterogeneity, source code availability, maintainability and complexity need to be considered [27].

Information systems must collaborate with each other in order to fulfil the requirements of stakeholders. The collaboration is achieved by integrating the systems. The integration at the system level can be done utilizing different interaction models, for example, integrated, bus-based, bridged, and coordinated [5].

Users require high quality and reliability. As a consequence, the information system needs a comprehensive inspection and testing process. Testing the interoperability and quality of an information system as a big bang in a deployment or introduction stage is difficult, costly or even impossible. The reasons of this are the heterogeneity, complexity, diversity of systems, and reuse of COTS, for which code is not available. For producing quality software the quality must be emphasized right at the beginning of the projects and the quality assurance must occur similarly while building and buying the components of the software system. For this reason, the software process should include high quality inspection and testing policies, which verify that all the functional and non-functional requirements of the stakeholders will be in the final product and the product does not have not-needed properties. Thus forward and backward traceability is required [4]. The requirements of stakeholders are gathered and then used, at the analysis level, in order to derive use cases and conceptual class hierarchy, which are then used as a starting point at the design phase [18,2].

Wu et al introduced that errors in component-based system can be inter-component faults, interoperability faults in system level, programming level, and specification level, and traditional faults [28]. They have also presented a maintenance technique for component-based software, where static analysis is used to identify the interfaces, events and dependence relationships that would be affected by the modifications. The results obtained from the static analysis are used to help test case selection in the maintenance phase. In this technique, they do not separate different dependency relationships.

We propose an improvement where dependencies inside one component do not interfere with external dependencies and present an effective and practical method for testing business component systems step by step. We consider functional requirements only. We utilize components from different granularity levels defined in Herzum and Sims [5]. The granularity hierarchy means that a business component system is a composition of business components, which in turn are compositions of lower level components. The advantages of this approach are the possibility to master development and deployment complexity, to decrease time to market, and to increase scalability of software systems. Also the great number of dependencies which occur in object-oriented approach [25] can be mastered better, because majority of the dependencies between classes remain inside one component where the number of classes is much less than in the total system or subsystem. The abstraction levels decrease the work needed in testing, because the testing work can be divided into sufficiently small concerns and the previously tested components can be considered as black boxes, whose test results are available. Furthermore, errors can be easily

detected, because not so many components are considered at one time.

The remainder of the paper is organized as follows: In Section 2 we describe the component properties and the component granularity levels. Our testing method in general is given in Section 3 and in Section 4 is the testing process for different granularity components. Related work is considered in Section 6. A conclusion is finally given in Section 7.

2. PROPERTIES OF COMPONENTS

2.1 Interfaces and Contracts

The definition of a component stresses its autonomy [5, 22]. Each component forms a cohesive and encapsulated set of operations (or services). A component should have minimal and explicitly defined relationships with other components minimizing coupling [13]. The semantics of the relationships should be defined as well as the syntax. This leads to the definition of interfaces of the component, which hide internal properties of the component. The *interface* of the component specifies the list of provided operations, for each operation the list of parameters, and for each parameter the type and direction (in or out). Interface Definition Language (IDL) [14] is usually utilized in the interface specification, where semantics can be described using preconditions, postconditions, and invariants [22].

However, components can not be isolated. A component should not be too wide and complicated. Thus it can not perform all the things by itself, but it collaborates with other components: First, a component may call other components by sending a message. In this paper, we consider only synchronous messaging. Thus the caller of the operation waits for the result. Each interface of the component has a dependency relationships of its own. Second, a component needs an execution environment, i.e. a socket, into which it plugs and within which it executes. The provided interfaces, the required dependencies for each interface and the specified execution environments of the component form a *contract* between a called component and a component caller [22]:

Contract = interface, dependencies, environment, specification.

A component can have several contracts. Different customers want different properties and at the maintenance stage version management can be solved by making new versions of the contracts. Similarly contracts are a profitable document between the producer of a component and customers, who want to buy the component. This contract can be utilized in build-time, too.

2.2 Granularity of Components

By partitioning a given problem space into a component form we utilize, in this research, the business component approach introduced by Herzum and Sims [5]. Components of different granularities, i.e. distributed component, business component, and business component system are described in the following chapters.

2.2.1 Distributed Component

The *lowest granularity* of software component is called a distributed component. A *distributed component* (DC) has a well-defined build-time and run-time interface, which may be network addressable. A DC can be deployed as a pluggable binary component to the socket given in the contract. Further a distributed component may have dependency relationships to other components. A DC can have one or more interfaces, which define the operations component offers and the parameters needed when calling the component. Thus an interface of DC can be defined as follows:

Interface = (operation, (parameter, type, [in/out]))^{*}

User interface implementation should be separated from business logic implementation and database access. This leads to the definition of categories for distributed components, i.e. user DC, workspace DC, enterprise DC and resource DC. User DC differs from the other DCs because it does not have network addressable interface but it has user interface [20]. In object-oriented approach the distributed component usually consists of classes and relationships between them, but traditional programming approaches can be used, too. Thus a DC hides the implementation details from the component user. Component technologies offer usually a degree of location transparency and platform and programming language neutrality. For example, a distributed component could be lab order entry.

2.2.2 Business Component

The *medium granularity* of software component is called a business component. A *business component* (BC) consists of all the artifacts necessary to represent, implement, and deploy a given business concept or business process as an autonomous, reusable element of a larger distributed information system. From the functional designer's point of view a business component can consist of user, workspace, enterprise and resource tiers. Each tier consists of zero or more DC whose category is the same as the tier. User and workspace tiers form a single-user domain. Enterprise and resource tiers form a multi-user domain.

In figure 1 a business component and the execution environment (CEE) for that component is presented. A distributed component can send a message to a component, which is at the same or at the lower tier as itself. Cyclic messaging should be avoided, because it violates normalization. Events, for example, error messages, such as database access violation, can go from lower tier to the upper tier. From the above follows that a business component is a composition component, whose parts are distributed components. The runtime interface of a BC is the union of all interfaces of its distributed components, which are visible outside the boundaries of the BC. A business component could be, for example, lab test, which could contain different tests and the structure of results.

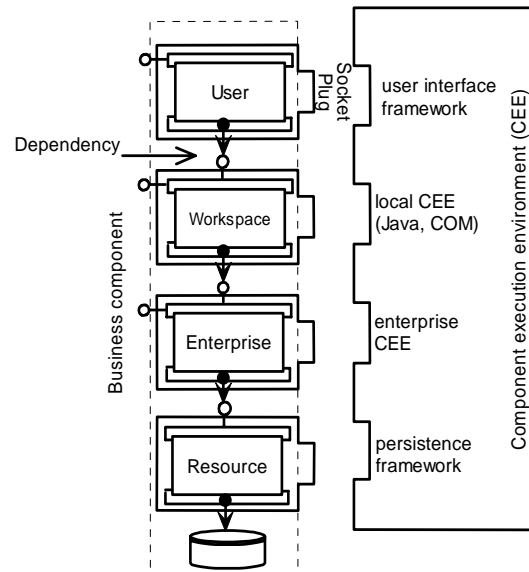


Figure 1. The business component and the component execution environment (Adapted from [5]).

2.2.3 Business Component System

The *largest granularity* of software components is called a business component system. A *business component system* (BCS) is a composition component, whose parts are business components that constitute a viable system. The runtime interface of BCS is the union of all interfaces of its business components, which are visible outside the boundaries of the BCS. Business components can be classified into functional layers, for example, process, entity, and utility. In figure 2 there is Lab test business component system. At the process layer there is one business component, Lab test workflow. At the entity layer there are Lab test, Test result analyzer, Department and Patient business components. Utility layer business components include Lab test codebook and Addressbook. Database integrity manager and Performance monitor are auxiliary business components. As before, messages can go from the upper level to the lower level. The granularity of components gives controllability to the testing process as can be seen in the following chapters.

3. TESTING PROCESS

3.1 Testing Method

In this paper we present a testing method, where components are tested from the lowest granularity to the highest granularity. Testing process is analogous at each level and the test results of lower levels are available while testing the upper levels. Thus the test manager can see the quality of components while planning the test of a composition component. It is a known fact in practice that software parts that contain most errors are risky areas for errors also in the future. The idea of the method is that at each level white box testing and black box testing occur alternately:

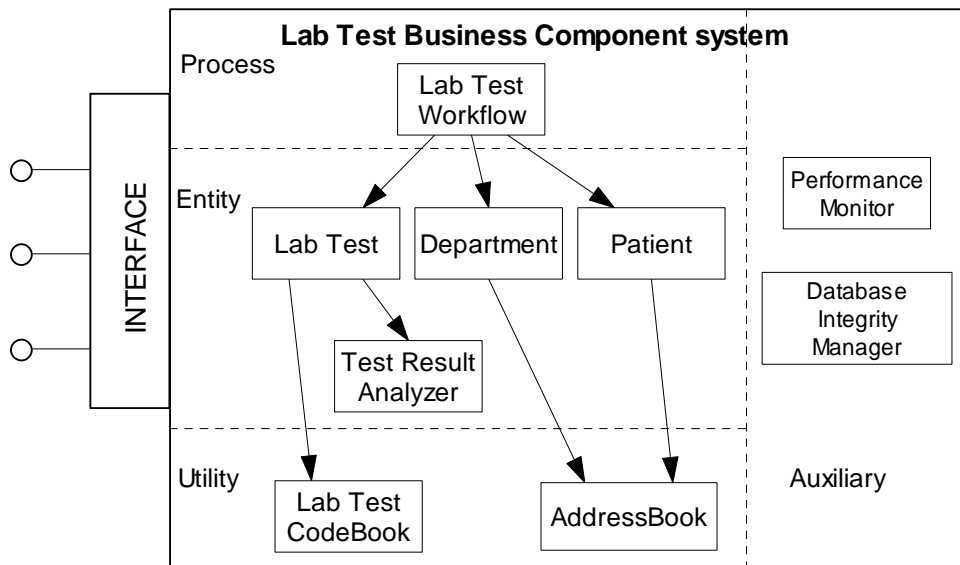


Figure 2. Lab Test Business Component System

- First, in unit testing phase the internal logic of the component is tested as a white box. Second, the external interface of the component is tested. Here the component is considered as a black box and it is tested in the execution environment, where it should work.
- In integration testing phase the component is considered as a composition component of its internal components. Now, the results of previous steps are available. First, the co-operation of internal components of the composition component is tested. Here, the internal components are black boxes. Second, the interface of the composition component is tested.

The alternation, presented above, is also true if we consider the role of component provider and component integrator:

The provider of the component needs white box testing for making sure that the component fulfils the properties defined in the contract and black box testing for making sure that the interface can be called in environments specified in contract.

The third-party integrator integrates self-made and bought components into a component-based system. He uses black box testing for making sure that he gets a product that fulfills requirements. Integration testing of self-made and bought components means that calling dependencies of components are considered. Thus from the systems point of view the internal logic of the system is considered. We denote this as white box testing although code of internal components is not available. At last external interfaces of the system are tested and here the total system is considered as a black box.

We can also consider a customer who buys a component-based system. He does not know the internal logic of the system. Thus he uses black box testing techniques for acceptance testing.

In the following chapters we first define test cases (chapter 3.2). When executing the system with carefully chosen test cases tester can see if the user requirements are fulfilled. Test cases are

derived from the use cases or contracts. Furthermore, we need a dependency graph (chapter 3.3), which shows if all the parts of the system have been covered by the test cases.

3.2 Test Cases

3.2.1 Definition of Test Cases

A test case is generally defined as input and output for the system under test. Kaner [7] describes that a test case has a reasonable probability of catching an error. It is not redundant. It's the best of its breed and it is neither too simple nor too complex. A test case is defined in Rational Unified Process [9] as a set of test inputs, execution conditions, and expected results developed for particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. We insert the granularity aspect in the definitions of test cases:

- Test case at the business component system level is an action flow between business components and users.
- Test case at the business component level is a sequence of operations between distributed components.
- Test case at the distributed component level is a method sequence between object classes.

3.2.2 Use Cases and Contracts

Use cases and scenarios provide means for communication between users and a software system [6, 21]. Thus they are useful while deriving the responsibilities of BCS. The responsibilities of total BCS, which is considered as a composition component, are divided into responsibilities of each internal BC of BCS. They are defined recursively if needed. Thus we can suppose that for BCS and BC we have use case descriptions, which show those needs of the stakeholders that are addressed to that component. With use cases we know how to actually use the system under test. Every

use case must have at least one test case for the most important and critical scenarios.

However, use case diagrams and scenarios usually show only the communication between users and a software system. Thus the co-operation of human actors is not presented. We propose that the definition of use case diagram is extended to contain human actions as well as automated actions [8]. Consequently, action flows can be derived from these use case diagrams and it is possible to test that human workflow fits together with actions of BCS.

Next, we present examples of test cases of different granularity components. Examples are simplified, but they describe how abstraction levels differ when moving from BCS level to DC level. At the DC level test cases are the most accurate and detailed.

For example, an action flow of Lab Test BCS (see Fig. 2) could be following (actors in parenthesis):

- Create or choose a patient; (human and Patient BC)
- Create a lab order; (human and Lab Test BC)
- Send the lab order to the lab; (human, Lab Test and Department BC)
- Take a sample; (human)
- Analyze the sample and return results; (Test Result Analyzer BC)
- Derive reference values; (Lab Test Codebook BC)
- Save lab test results and reference values; (Lab Test BC)

An operation sequence of Lab Test BC could be following:

- Input a patient number; (human and user DC)
- Find lab test results with reference values; (enterprise and resource DC)
- Output lab test results with reference values; (user DC)
- Evaluate results; (human)
- Decide further actions; (human)
- Send lab test results and advice for further actions to the department; (human, user and enterprise DC)

A method sequence of Lab Test user DC could be following:

- Input a patient number; (human and Patient class)
- Choose an activity lab test consulting; (human and Menu class)
- Send message “find lab test results” to enterprise DC; (proxy)
- Receive lab test results; (proxy)
- Output lab test results; (Lab Test class)
- Output reference values; (Reference class)

The other possibility to define test cases is to utilize contracts, specifying the operations offered by the components. In testing we must have a requirement document in which each operation of the interfaces has been described in detail. There we get input parameters of the operations and their domains. For each

operation each parameter's input domain is divided in so called equivalence classes [15]. Equivalence partitioning is always the tester's subjective view and thus may be imperfect. If the operation has many input parameters then the equivalence classes for the whole input can be designed as a combination of the equivalence classes. This leads to an explosion in the number of equivalence classes. Test cases are selected so that at least one test input is selected from every equivalence class. So we get testing which is on one hand effective and covers customers' requirements and on the other hand it is not too arduous and complex. Redundancy is also as minimal as possible. For distributed components in resource, enterprise and workspace tier contracts may be the only possibility to define test cases. But it should be remembered that contracts do not specify the cooperation between more than two components. Thus they are not sufficient while testing action flow of stakeholders and the usability of the whole system.

Finally we check that test cases traverse all the paths of all the dependency graphs as presented in the next chapter.

3.3 Dependency Graph

3.3.1 General

Test cases, which are defined by use cases or contracts do not necessarily cover the whole functionality of the system. Besides use cases and contracts, we also need the dependency graph, which shows dependencies between the components of the same granularity level. If we do not know all the dependencies we do not know how to test them. We use the dependency graph to assure that the whole functionality of the component has been covered by test cases. Without dependency graph there may remain some critical paths that have not been executed or there may be some paths that have been tested many times. Redundant testing increases always the testing costs. Then test cases have to be removed. If there are paths, which are not traversed at all we must examine carefully if the components on those paths are needless for some reason or we have to add test cases so that all the paths will be traversed.

The following chapter describes how to compose the dependency graph and provides an example. The algorithm creates a dependency graph for each external interface of the composition component. So testing and maintaining component-based systems is easier than if we had only one big graph.

3.3.2 Dependency Graph Creation

We create a graph based on dependencies between different components in a composition component. Term composition component can either be a business component or a business component system. If it is a business component, dependencies are between distributed components and if it is a business component system, dependencies are between business components. A node in a graph represents a component. A directed edge from component A to component B means that component A depends on component B (component B offers functionality to component A). The inner for-loop checks dependencies only at the same tier or tiers below because messages from one component to another go from upper to lower tier. The outer for-loop checks all the external interfaces the

composition component has and creates a graph for each interface. Our algorithm follows the breadth first search algorithm.

Algorithm: Creates composition component dependency graphs.

Input: A composition component and its contracts.

Output: Composition component dependency graphs.

Variables:

- A set *called* includes all the components that have been traversed.
- A set *not_visited* includes components that have not been yet traversed.
- A set *targets* includes components that have already been as target components.

```

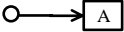
for each external interface of a composition component {
  start with the component  $c_0$  to which the interface refers;
  create an edge from the interface to  $c_0$ ;
  not_visited = {all components in the composition
  component};
  called =  $\{c_0\}$ ;
  targets =  $\emptyset$ ;
  while not_visited  $\neq \emptyset$  and called - targets  $\neq \emptyset$  {
    select target component from the (called - targets) set;
    for each component c in the composition component in
    the same tier or below than target {
      if target depends on a component c then
        called = called  $\cup c$ ;
        create an edge from target to c;
      }
    not_visited = not_visited - target;
    targets = targets  $\cup$  target;
  }
}

```

If BCS includes components, which can not be reached by any interface, those components do not belong to the graph. If there are cycles in the graph the algorithm reveals them as can be seen in the following example.

3.3.3 Example

In this chapter there is an example of using the previous algorithm. In figure 3 there are distributed components A - F and their dependencies in a 3-tiered business component. Let's start with the user level external interface. It refers to the component A.

 // create an edge from the interface to c_0
 not_visited = {A,B,C,D,E,F}

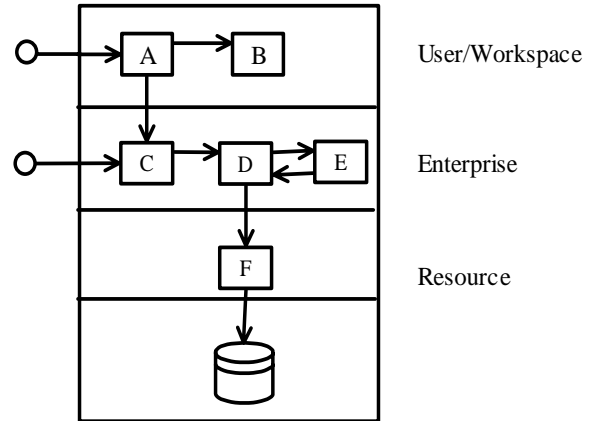
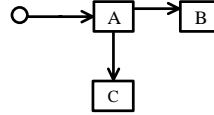


Figure 3. Graphical view of distributed components in a business component

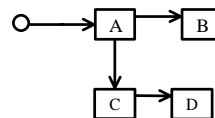
called = {A}
 targets = \emptyset
 not_visited $\neq \emptyset$ and called - targets = {A} - \emptyset = {A}
 target = {A}
 called = {A} \cup B = {A,B}



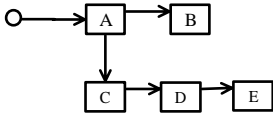
called = {A,B} \cup {C} = {A,B,C}



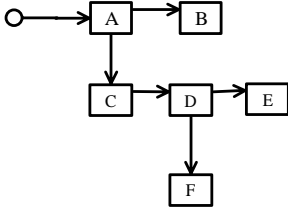
component A has no other dependencies
 not_visited = {A,B,C,D,E,F} - {A} = {B,C,D,E,F}
 targets = $\emptyset \cup$ {A} = {A}
 not_visited $\neq \emptyset$ and called - targets = {A,B,C} - {A} = {B,C}
 target = B
 no dependencies
 not_visited = {B,C,D,E,F} - {B} = {C,D,E,F}
 targets = {A} \cup B = {A,B}
 not_visited $\neq \emptyset$ and called - targets = {A,B,C} - {A,B} = {C}
 target = C
 called = {A,B,C} \cup D = {A,B,C,D}



no other dependencies
 not_visited = {C,D,E,F} - {C} = {D,E,F}
 targets = {A,B} \cup C = {A,B,C}
 not_visited $\neq \emptyset$ and called - targets = {A,B,C,D} - {A,B,C} = {D}
 target = {D}
 called = {A,B,C,D} \cup E = {A,B,C,D,E}



called = {A,B,C,D,E} \cup F = {A,B,C,D,E,F}



no other dependencies

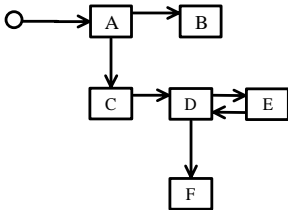
not_visited = {D,E,F} - {D} = {E,F}

targets = {A,B,C} \cup D = {A,B,C,D}

not_visited $\neq \emptyset$ and called - targets = {A,B,C,D,E,F} - {A,B,C,D} = {E,F}

target = {E}

called = {A,B,C,D,E,F} \cup D = {A,B,C,D,E,F}



no other dependencies

not_visited = {E,F} - {E} = {F}

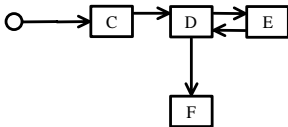
targets = {A,B,C,D} \cup E = {A,B,C,D,E}

not_visited $\neq \emptyset$ and called - targets = {A,B,C,D,E,F} - {A,B,C,D,E} = {F}

target = {F}

no dependencies

Next the outer for-loop examines other external interfaces; in this case enterprise level interface, which refers to component C. Algorithm creates the following graph for this interface.



3.3.4 Selecting Test Cases

When we have created dependency graphs we have to create test cases based on those graphs. Test cases are created so that as many paths in a graph as possible are covered by one test case. This is called path coverage. Test suite satisfies 100% path coverage if all the paths in the dependency graphs have been executed. We should remember that the graphs are not very large because of components' granularity and because interfaces act as centralized connection points. So the complexity is lower if

compared to the graphs of traditional or object-oriented software. We will study automatic test case selection in the future research.

4. TESTING COMPONENTS OF DIFFERENT GRANULARITIES

4.1 Distributed Components

Technical heterogeneity means use of different component technologies, programming languages and operating system environments. Productivity and flexibility of software implementation is increased by separating the functional logic from the runtime environment (socket) and the dependencies of the component technology, i.e. interface implementation and proxies which implement dependencies [5]. This profits the testing process, too. The functional logic is tested separately from interface implementation and proxies, which are substituted with driver and stub correspondingly if needed. Testing a distributed component depends on implementation. If a DC has been implemented by traditional techniques we can use traditional testing techniques and if it has been implemented by object oriented techniques we can use object oriented testing methods. In object oriented approach the functional code is usually implemented with classes and relationships between them. Testing means that methods and attributes of each class must be tested as well as the inheritance relationship between classes and association and aggregation relationships between objects. At the DC level test cases are usually derived from contracts. The execution of an operation, defined in the contract of DC, causes usually collaboration between several objects, which communicate with each other by sending messages. Thus the method sequence is one important subject to be tested. The objects dependency graphs can be derived analogously to the method presented in chapter 3.3 and it should be consistent with UML's collaboration diagrams defined in analysis stage. Furthermore, when we are testing a DC implemented by object oriented methods we can use several testing techniques [11]. For object state testing there are methods like [12, 23] and for testing inheritance relationship between classes there are methods like [16, 10].

Interfaces of DC must be tested locally and from the network. For user DCs the usability testing is important as well as the functionality tests. Testing resource tier DCs is more difficult if several databases or independent islands of data [5] are used.

4.2 Business Components

Vitharana & Jain [26] have presented a component assembly and testing strategy:

"Select a set of components that has maximum number of calls to other components within the set and minimum number of calls to other components outside the current set.

Of the remaining components, select a component (or a set of components) that has the maximum number of interactions with methods in the current application subsystem."

The strategy has been illustrated by an example but it has not been proved. However, the authors critique the strategy: The logical relationships between components should be taken into consideration while developing an integration test strategy.

We propose that in assembly and testing the business logic should be taken into account. Business components form a coherent set of properties, thus to test them as a whole is worthwhile. A business component is integrated from distributed components. Thus testing business component means:

- First, the integration testing of those distributed components, which belong to the business component is performed.
- Second, the external interface of the business component is tested.

While integrating a BC we propose that the integration strategy by Vitharana and Jain is modified as follows:

The assembly and testing go in two parallel parts:

In single-user domain part, the user and workspace tiers are integrated:

- It is profitable to start integration from user tier. Consequently, the comments from stakeholders are received as soon as possible.
- The workspace tier should be integrated next, because it is connected with the user tier.

In multi-user domain part, the resource and enterprise tiers are integrated:

- The resource tier is integrated first, because the DC in resource tier does not send messages to any lower tier.
- The enterprise tier is integrated next because it sends most messages to the resource tier.

Finally the total BC is integrated by combining the results of the above two parts. The above approach has several advantages. For example time to market decreases and controllability increases.

Testing business components is divided into two phases:

Phase 1:

BC's internal logic is considered using dependency graph similarly as before. Here each DC, which belongs to the business component is a black box, but BC itself is considered as a white box. Interfaces and dependencies are tested. The dependency graph is generated using the algorithm presented in chapter 3.3. If some of the DCs is not ready and has not passed through unit testing it is substituted with a stub.

The best way to derive test cases for BC is to utilize use cases. Because BC is a coherent set of operations of the business concept it is plausible that the most important and critical use cases of BC are defined at the analysis stage. Thus test cases can be built according to these use cases. The distributed components in user tier are the only components for which the user gives the input. For other BC-external interfaces the inputs come from some other systems or from the network. The values needed in BC's internal dependencies are calculated inside DCs. Normal cases are tested before exceptions [21]. While considering exceptions the events go from the lower level to the upper level. For example, when an exception is noticed, a resource DC sends an event to an enterprise DC, which further sends an event to a user DC. This means that the algorithm forming dependency graph needs to be slightly modified while testing exceptions.

If use cases are not available, the contracts of the distributed components, which are visible outside the boundaries of BC are used. In this case the designer should decide the order of the operations.

Phase 2:

BC's external interface, especially the network addressability, should be tested. Here the BC is a black box. Partly the same test cases as before can be used. Now the internal logic is not considered, but the correspondence of operation calls with input parameter values is compared to the return values. Thus all the contracts of BC must be tested.

4.3 Business Component Systems

Business component system is assembled and tested using strategy presented in [26]. Utilities are often ready-made COTS, which need only acceptance testing. This means that the interface is tested and the component is seen as a black box. Entity BCs call utilities, thus entities are tested next. Process BCs call entities, thus they are tested last. Thus the order is: first utility BC, second entity BC, third process BC.

Testing business component system means:

- First, integration of those business components, which belong to the business component system is tested.
- Second, the external interface of business component system is tested.

In integration testing BCS's internal logic is considered. Here each BC of BCS is a black box, which has been unit tested. BCS itself is considered as a white box. Interfaces and dependencies are tested utilizing dependency graph similarly as before.

Test cases of BCS are constructed using use cases, which show the action flow of users of the BCS. Of course, it is possible that the inputs come from some other system, but these are considered similarly to human inputs.

It is sufficient to test that the most important and critical action flows of users go through without errors and that BCs call each other with right operations and parameters. Exception cases [21] are tested after normal cases. While considering exceptions an entity BC can send events to a process BC. This means that forming dependency graph needs to be slightly modified. The contracts of components specify what the components offer, not what the users need and not the order of users actions, thus contracts of BCs are not useful while testing BCS. At last, BCS's external interface is tested with local calls and calls over network. Here all the contracts of BCS must be tested.

In conclusion, before testing the whole business component system each business component in it is tested. Before a business component is tested each distributed component in it is tested. The dependencies considered stay all the time at one component granularity level.

5. RELATED WORK

According to Weyuker's [24] anticomposition axiom adequately testing each individual component in isolation does not necessarily suffice to adequately test the entire program. Interaction cannot be tested in isolation. So, it is important to test

components' interaction in the new environment as components are grouped together.

Our work has got influence from Wu et al. [28]. However, we wanted that dependencies stay at the same abstraction level, i.e. they must not go from upper level to the lower level or vice versa in testing. In presentation of Wu et al. there is no clear separation of abstraction levels. For example, a dependency between components causes dependencies between classes. The interface implementation and functional logic are tested separately and not in order. In our algorithm, dependencies stay at the same granularity component level: in system level, in business component level, or in distributed component level. The dependencies between classes in object oriented systems need to be considered only at the lowest level. This reduces the dependencies and especially those dependencies, which must be considered at the same time.

Regnell et al. have considered use case modeling and scenarios in usage-based testing [17]. They investigate usage of software services in different users and users' subtypes. They consider dependency relationships, where relationships go from component level to the class and object level as in [28].

Gao et al. have presented Java-framework and tracking model to allow engineers to add tracking capability into components in component-based software [3]. The method is used for monitoring and checking various behaviors, status performance, and interactions of components. It seems that the results of Gao et al. could be added in our approach in order to support the debugging aspects.

Our research emphasizes testing functional requirements of business component system. The quality requirements of stakeholders such as security, response times and maintainability must be tested too. This has been considered in [1]. Different quality properties need to be tested separately although scenarios can be utilized here too. Testing quality requirements leads to the consideration of architectures.

6. CONCLUSION

We have presented a method for testing functionality of business component systems. For testing functionality we utilize test cases and dependency graphs. Test cases are derived from use cases or contracts.

Why do we need test cases and dependency graphs? To assure that the whole composition component's functionality has been covered by test cases it is necessary to use the dependency graphs. If we only test that the test cases are executed right, they give right result, and leave the system in consistent state, there may remain some critical paths in the system that have not been executed or there may be some paths that have been tested many times. If there are paths, which are not traversed at all our test suite does not correspond the functionality of the system. In this case, we must examine carefully if

- new test cases should be inserted or
- the components on non-traversed path are needless for some reason.

In our method, components of different granularities are tested level by level. Thus in integration testing the dependencies stay

inside a business component system at the business component level. While testing business components the dependencies stay at the distributed component level. At distributed component level we consider the dependencies between classes. From the above follows that dependencies stay simple and at the same level, and the dependencies tested at the same time are similar, except at the DC level. Thus testing work is divided into small pieces and the amount of testing work decreases. This facilitates regression testing too.

Our work has been done at the University of Kuopio as a part of PlugIT research project in which our testing method is being evaluated in practice. The validation of the method containing also theoretical proof of decreasing the work in testing in practice is going on at the moment. The goal of PlugIT project is to reduce threshold of introduction of health care information systems by defining more effective and open standard solutions for system level integration. Our concern is for the quality assurance and testing of health care information systems.

7. ACKNOWLEDGMENTS

We would like to thank Hannele Toroi, testing manager at Deio for giving us insight into test implementation and testing problems in practice. This work is part of PlugIT project, which is funded by the National Technology Agency of Finland, TEKES together with a consortium of software companies and hospitals.

8. REFERENCES

- [1] Bosch, J. Design and use of software architectures. Adopting and evolving a product-line approach. Addison-Wesley, 2000.
- [2] Fowler, M., and Kendall, S. UML Distilled Applying the standard Object Modeling Language. Addison-Wesley, 1997.
- [3] Gao, J., Zhu, E., Shim, S., and Chang, L.: Monitoring software components and component-based software. In Proc. of 24th Annual International Computer Software & Applications Conference, 2000.
- [4] Gotel, O. Contribution structures for requirements traceability. PhD thesis, University of London, 1995. <http://www.soi.city.ac.uk/~olly/work.html>.
- [5] Herzum P., and Sims, O. Business Component Factory. Wiley Computer Publishing, New York, 2000.
- [6] Jacobson, Christerson, M., Jonsson, P., and Övergaard, G. Object-Oriented Software Engineering – A Use Case Driven Approach. Addison-Wesley, Harlow, 1995 2nd edn.
- [7] Kaner, C. Testing computer software. John Wiley & Sons, New York, 1999.
- [8] Korpela, M., Eerola, A., Mursu, A., and Soriyan, HA.: Use cases as actions within activities: Bridging the gap between information systems development and software engineering. Abstract. In 2nd Nordic-Baltic Conference on Activity Theory and Sociocultural Research, Ronneby, Sweden, 7-9 September 2001.

- [9] Kruchten, P. The Rational Unified process, an introduction. Addison-Wesley, 2001.
- [10] Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., and Chen C.: Change impact identification in object oriented maintenance. in Proc of IEEE International Conference on Software Maintenance 1994, 202-211.
- [11] Kung, D., Hsia, P., and Gao, J. Testing object-oriented software. IEEE computer society, USA, 1998.
- [12] Kung, D., Lu, Y., Venugopalan, N., Hsia, P., Toyoshima, Y., Chen C., and Gao, J.: Object state testing and fault analysis for reliable software systems. In Proc. of 7th International Symposium on Software Reliability Engineering, 1996.
- [13] Meyer, B. Object-oriented software construction. Prentice Hall, London, 1988.
- [14] Mowbray, T., and Ruh, W. Inside CORBA: Distributed object standards and applications. Addison-Wesley, 1997.
- [15] Myers, G. The art of software testing. John Wiley & Sons, New York, 1979.
- [16] Perry, D., and Kaiser, G.: Adequate testing and object oriented programming. Journal of Object-Oriented Programming, Jan/Feb, 1990, 13-19.
- [17] Regnell, B., Runeson, P., and Wohlin, C.: Towards integration of use case modelling and usage-based testing. The Journal of Systems and Software, 50, 2000, 117-130.
- [18] Robertson, S., and Robertson, J. Mastering the requirements process. Addison-Wesley, 1999.
- [19] Roper, M. Software testing. McGraw-Hill, England, 1994.
- [20] Sametinger, J. Software Engineering with Reusable Components. Springer-Verlag, 1997.
- [21] Schneider, G., and Winters, J.P. Applying use cases. Addison-Wesley Longman, 1998.
- [22] Szyperski, C. Component Software: Beyond Object-Oriented Programming. Addison-Wesley, Harlow, 1999.
- [23] Turner, C.D., and Robson, D.J.: The state-based testing of object-oriented programs. In Proc. of IEEE Conference on Software Maintenance 1993, 302-310.
- [24] Weyuker, E.: The evaluation of program-based software test data adequacy criteria. Communications of the ACM, 31:6, June 1988, 668-675.
- [25] Wilde, N., and Huitt, R.: Maintenance Support for Object-Oriented Programs. IEEE Transactions on Software Engineering, 18, 12, Dec. 1992, 1038-1044.
- [26] Vitharana, P., and Jain, H.: Research issues in testing business components. Information & Management, 37, 2000, 297-309.
- [27] Wu, Y., Pan, D., and Chen, M-H.: Techniques for testing component-based software. Technical Report TR00-02, State University of New York at Albany, 2000.
- [28] Wu, Y., Pan, D. and Chen, M-H. Techniques of maintaining evolving component-based software. In Proceedings of the International Conference on Software Maintenance, San Jose, CA (USA), October 2000

Sample Test Plan

TehoTest

Version 1.0

11/Jan/2002

REVISION HISTORY

Version	Date	Authors	State (Changes, updates)
1.0	11/Jan/2002	Marko J	Original

Content

1	INTRODUCTION	4
1.1	PURPOSE.....	4
1.2	BACKGROUND	4
1.3	SYSTEM OVERVIEW	4
1.4	REFERENCES	4
2	TEST ENVIRONMENT	5
2.1	HARDWARE	5
2.2	SOFTWARE.....	5
3	STAFF AND TRAINING NEEDS	5
4	ROLES AND RESPONSIBILITIES.....	5
4.1	MANAGEMENT TEAM	5
4.2	TESTING TEAM	5
4.3	TESTING SUPPORT TEAM	5
4.4	PROJECT ROLES AND RESPONSIBILITIES	6
5	RESULTS AND DELIVERABLE DOCUMENTS.....	6
6	FEATURES TO BE TESTED	6
6.1	USE CASE 1.....	6
6.2	USE CASE 2.....	6
6.3	USE CASE 3.....	6
7	FEATURES NOT TO BE TESTED.....	7
7.1	USE CASE 4.....	7
8	TEST SCOPE.....	7
8.1	FUNCTIONAL TESTING	7
8.1.1	User Interface Testing/Usability	7
8.2	QUALITY / NON-FUNCTIONAL TESTING.....	7
8.2.1	Performance Testing	7
8.2.2	Load Testing	7
8.2.3	Security Testing	8
8.2.4	Data and Database Integrity Testing	8
8.2.4	Recovery Testing	8
8.3	CONFIGURATION TESTING / PRODUCT MANAGEMENT.....	8
8.4	INSTALLATION TESTING.....	8
9	PROJECT MILESTONES	9
10	RISKS	9
11	APPROVALS.....	9

1 Introduction

1.1 Purpose

This document describes the plan for testing the **TehoTest** System. This test plan identifies existing project information, resources for testing and functions to be tested. Additionally it describes testing strategies, provides an estimate of the test efforts. It includes a list of the deliverable test documentation and risks, project milestones and approvals.

1.2 Background

The TehoTest project was started on 1st of October in order to develop communication and data exchange in health care centres and hospitals. There have been several studies concerning data exchange between decentralized objects in health care sector.

1.3 System Overview

TehoTest is a Java based application for inserting information about patient to the JDBC database.

1.4 References

This test plan includes references to following project documents:

- Project Management Plan 1.0
- Requirements Specification Document 1.0
- System Design Document 1.0
- Product Description

2 Test Environment

2.1 Hardware

Hardware requirements for testing are

- PentiumIII processor
- 300MB free memory
- CD-ROM drive
- client and server computer
- HP LaserJet printer

2.2 Software

Software requirements for testing are

- MS Office
- Windows 2000
- Rational TestStudio
- Rational AnalystStudio
- JDK 1.3
- Jbuilder 5.0

3 Staff and Training Needs

Worker	Need for Training	Extra Information
Marko	PerformanceStudio	Training 16 March
David	Purify	
Samuel	Quantify	

4 Roles and Responsibilities

4.1 Management Team

Management team for testing consists of

- Test Manager Tim Tester, tim@hotmail.com
- Project Manager Jim Beam, beam@hotmail.com

4.2 Testing Team

Testing team consists of

- Software Designer Tim Tester, david@hotmail.com
- Programmer Samuel System, samuel@hotmail.com

4.3 Testing Support Team

There is no testing support team within this project.

4.4 Project Roles and Responsibilities

Role	Worker	Responsibility
Test Manager	Tim Tester	<ul style="list-style-type: none">• Resource planning• Reporting to the management
Test Designer	David Designer	<ul style="list-style-type: none">• Identifying, prioritizing and implementing of test cases.• Produce test plan and test model
System Tester	Samuel System	<ul style="list-style-type: none">• Execute tests• Log results• Error recovery• Document defects

5 Results and Deliverable Documents

Deliverable Document	Responsible Worker	Review	Due Date
Test Plan	Marko J	Project Manager/Customer	
Test Environment	David	Project Manager	
Test Cases	David	Project Manager/Customer	
Test-Incident Report	Marko J	Project Manager	
Test Summary Report	David	Project Manager/Customer	

6 Features to be Tested

6.1 Use Case 1

UC1 Login to the system.

6.2 Use Case 2

UC2 Insert patient's information

6.3 Use Case 3

UC3 Logout from system

7 Features Not to be Tested

7.1 Use Case 4

UC4 (Insert medical data about patient) won't be tested in this version.

8 Test Scope

The aim of TEHO test program is to verify that the TEHO system satisfies the requirements specified in System Requirements Specification. Automated test tools will be used to help testing process when it is possible.

8.1 Functional Testing

8.1.1 User Interface Testing/Usability

Test Objective	Navigation through the application (windows, fields)
Technique	
Completion Criteria	
Special Considerations	

8.2 Quality / Non-Functional Testing

8.2.1 Performance Testing

Test Objective	System response when 30 user accesses to the system
Technique	
Completion Criteria	
Special Considerations	

8.2.2 Load Testing

Test Objective	Workload for the application
Technique	
Completion Criteria	
Special Considerations	

8.2.3 Security Testing

Test Objective	Non-users cannot access the system
Technique	
Completion Criteria	
Special Considerations	

8.2.4 Data and Database Integrity Testing

Test Objective	Access to the database
Technique	
Completion Criteria	
Special Considerations	

8.2.4 Recovery Testing

Test Objective	Recovery from system failures (system boot)
Technique	
Completion Criteria	
Special Considerations	

8.3 Configuration Testing / Product Management

Test Objective	Functionality using Microsoft Internet Explorer V5.x and Netscape V4.x
Technique	
Completion Criteria	
Special Considerations	

8.4 Installation Testing

Test Objective	Installation Program for Teho Application
Technique	
Completion Criteria	
Special Considerations	

9 Project Milestones

Task	Effort	Start Date	End Date
Plan Test			
Design Test			
Implement Test			
Execute Test			
Evaluate Test			

10 Risks

Number	Risk	Effect	Mitigation Strategy
1	Load testing will be delayed if the maximum load for the system cannot be created	2 weeks schedule slip	Program for load testing has been ordered from XXX Ltd
2	Installation program is delayed		

11 Approvals

The persons who must approve this plan are listed as follows:

- Project Manager (XX Ltd)
- Project Manager (YY Ltd)
- Project Management Group
 - Sam Software
 - Tim Dollar
 - Frank McRobin

How to derive test cases from user requirements

If you have any comments or feedback concerning this document, please send mail to: mjantti@hytti.uku.fi

1 Introduction

This example describes how test cases can be derived from requirements specification documents with a *black box* testing method including three approach models: testing of the application with one dialog, testing of the application with many dialogs and testing of the multi-user system.

Sample applications in this document are based on the requirements specification that is shown in the second chapter. Chapter 3 describes the test case design for an application with one dialog using equivalence partitioning as a testing method. Chapter 4 introduces the test case design for an application with many dialogs. The topic of the chapter 5 is how to test action flows. The last chapter includes test case reports for sample applications of this document.

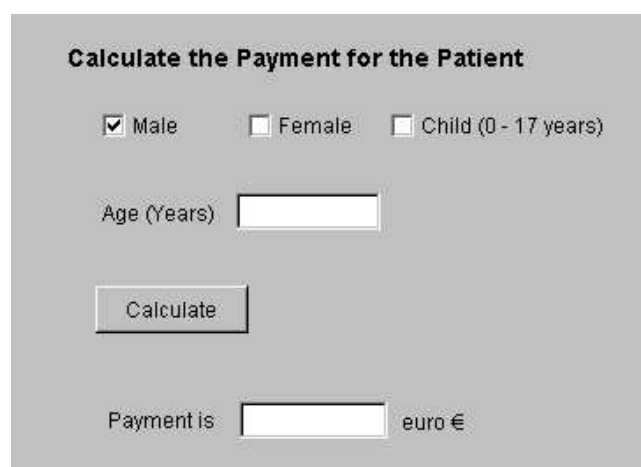
2 Requirements Specification

Patient System consists of two applications:

- Calculate Payment application
- Patient Management application

Both applications include a graphical user interface. The system is based on a multi-user database.

2.1 Calculate Payment application



Calculate the Payment for the Patient

☒ Male ☐ Female ☐ Child (0 - 17 years)

Age (Years)

Payment is euro €

Figure 1: Calculate Payment application

Requirement 1: Patient Type

User is able to choose patient type (male, female, child)

Requirement 2: Patient age

User is able to enter patient age as a positive integer value.

Requirement 3: Payment

Payment is calculated based on age and type of the patient. The result will be displayed on the screen.

Male

Age	Payment
18 - 35	100 euro
36 - 50	120 euro
51 -	140 euro

Female (* Payments in this example are not same than in a real world)

Age	Payment
18 - 30	80 euro
31 - 50	110 euro
51-	140 euro

Child

Age	Payment
0 - 17	50 euro

2.2 Patient Management application

Requirement 4: Insert patient

The user is able to insert patient data in the patient database (name, personal id, adress and symptoms).

Requirement 5: Search patient

The user is able to search patient data with keywords.

Requirement 6: Update patient

The user must be able to update patient data. Results are displayed on screen.

Requirement 7: Close the program

The user is able to close the program.

3 Test case design for an application with one dialog

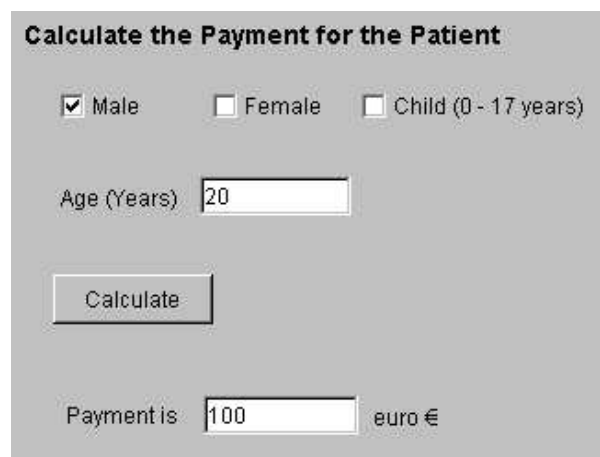
3.1 Equivalence Partitioning

All input fields in the application must be tested. However, the system cannot be tested with all possible inputs because it requires too much time and work. The set of input values for the system must be partitioned for example by using a test case selection like *equivalence partitioning*.

For each input parameter we can derive equivalence classes from requirements specification. Each equivalence class should represent a main processing path of input values through the system.

Test cases should also be created for boundary values, special characters and decimal values. The system should give for user warnings of illegal inputs (Error Messages). The following example shows how equivalence classes and test cases can be created for a simple application that calculates patient payments (Figure 2).

At first the user of this application chooses patient type in checkboxes (Male, Female, Child). Checkboxes cannot be left empty which reduces the possibility to give incorrect input values (default value is Male). Age should be given as whole years (positive integer). By pressing Calculate button application calculates the payment based on information about the patient type and age.



Calculate the Payment for the Patient

☒ Male ☐ Female ☐ Child (0 - 17 years)

Age (Years)

Payment is: euro €

Figure 2:User has filled data fields in Calculate Payment application

3.2 Equivalence classes for Calculate Payment application

There are two input parameters that user can enter in Calculate Payment application: patient type and age.

Patient Type

- Male
- Female
- Child

Age

- < 0 (invalid)
- 0-17 (valid)
- 18-35 (valid)
- 36-50 (valid)
- 51-145 (valid)
- > 145 (invalid)
- not integer (illegal)

3.3 Test Cases for Calculate Payment Application

Following tables consist of designed test cases. Three first columns include input values that user has given to the system. The last column *Expected results* shows the end state or message from the system. Expected results are derived from Requirements Specification. If a user has entered an invalid input value (for example negative values), the system returns an error message.

Male	Age		Expected Result
Choose Male	-1	Press Calculate	Error Message
Choose Male	5	Press Calculate	Error Message
Choose Male	18	Press Calculate	100 euro
Choose Male	36	Press Calculate	120 euro
Choose Male	51	Press Calculate	140 euro
Choose Male	146	Press Calculate	Error Message
Choose Male	@#lll	Press Calculate	Error Message

Female	Age		Expected Result
Choose Female	-5	Press Calculate	Error message
Choose Female	0	Press Calculate	Error message
Choose Female	18	Press Calculate	80 euro
Choose Female	31	Press Calculate	110 euro
Choose Female	51	Press Calculate	140 euro
Choose Female	146	Press Calculate	Error Message
Choose Female	aWqi	Press Calculate	Error Message

Child	Age		Expected Result
Choose Child	-45	Press Calculate	Error Message
Choose Child	15	Press Calculate	50 euro
Choose Child	18	Press Calculate	50 euro
Choose Child	40	Press Calculate	Error Message
Choose Child	jxxcc	Press Calculate	Error Message
Choose Child	146	Press Calculate	Error Message
Choose Child	!#%	Press Calculate	Error Message

4 Test Case Design for an Application with many Dialogs

Test case design for an application with many dialogs in a sequence needs more time and effort than designing simple test cases. A simple way to describe the procedure of test case is to refer UML (Unified Modelling Language) diagrams like use case and activity diagrams. All dialogs of the application and UML diagrams as follows are usually included in system specification documents.

4.1 Use Case Diagram

The use case diagram shows the interaction between the application and the user. The following picture (Figure 3) is a use case diagram of Patient Management application for inserting patient data. The *actors* of this use case are *nurse* and *patient system*. Use cases are for example *Insert patient* and *Update patient*.

Insert patient includes for example *Save patient* use case. A nurse is able to insert patient (fill name, personal id, address, symptoms and save them into the database) and update patient data. The nurse can also search a patient from the database or close the program.

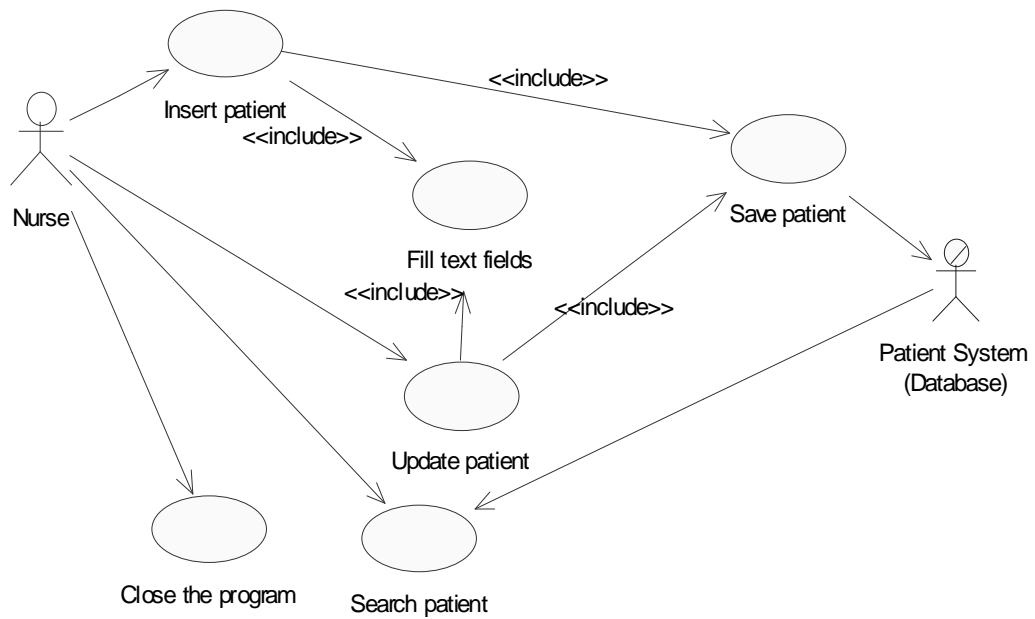


Figure 3: A Use Case Diagram of Patient Management application

4.2 Activity Diagram

The following activity diagram (Figure 4) describes the work flow and the order of displayed dialogs of Patient Management application. Arrows represent transitions from one function to the another. When a user has logged in, the main dialog of the program is displayed for him/her. The user fills all text fields (name, personal id, address, symptoms) and presses Save button.

If text fields contain invalid data, system displays an error message (incorrect inputs) and the user can return to the main dialog (Ok button) to check values in data fields. If there were no errors, system asks whether the user would like to save data. Cancel button returns the user back to the main dialog without saving data and Yes button saves data into the database. The user receives a message while data was successfully saved.

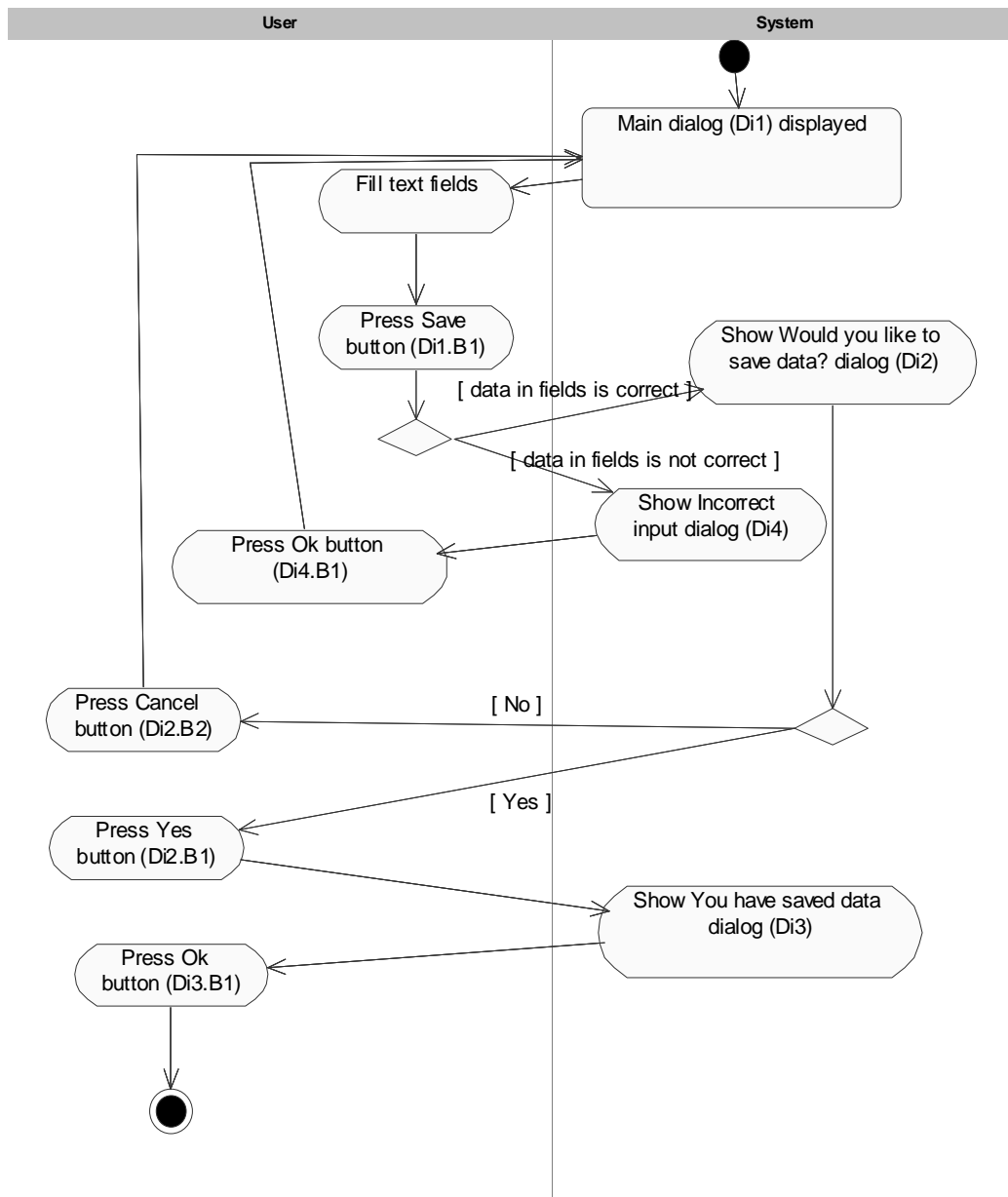
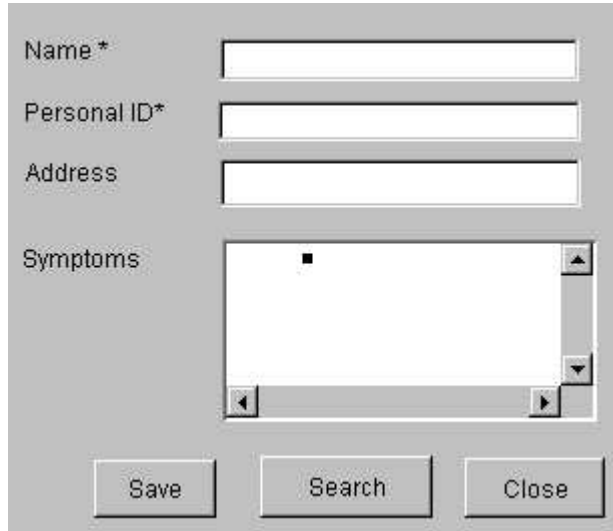


Figure 4: An activity diagram of Patient Management application

4.3 Dialogs of the Patient Management application

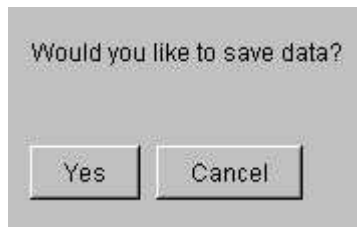
Patient Management application consists of following dialogs:



The Main Dialog (Di1) is a window with a light gray background. It contains four input fields stacked vertically on the left side, each with a label to its left: 'Name *', 'Personal ID*', 'Address', and 'Symptoms'. The 'Name *' and 'Personal ID*' labels are in bold. Each label is followed by a white rectangular text box. The 'Symptoms' text box is larger and has a small black square cursor in the top-left corner. To the right of the text boxes are three buttons: 'Save', 'Search', and 'Close', arranged horizontally. The buttons have a light gray background and a thin black border.

Figure 5: Main Dialog (Di1)

Notification Dialogs



The Notification Dialog (Di2) is a small window with a light gray background. It contains a single line of text: 'Would you like to save data?'. Below the text are two buttons: 'Yes' and 'Cancel', arranged horizontally. The buttons have a light gray background and a thin black border.

Figure 6: Would you like to save data – dialog (Di2)



The Notification Dialog (Di3) is a small window with a light gray background. It contains a single line of text: 'You have saved data'. Below the text is a single button: 'Ok'. The button has a light gray background and a thin black border.

Figure 7: You have saved data – dialog (Di3)

Error Message Dialogs



Figure 8: Incorrect input – dialog (Di4)

4.4 Test Cases for Patient Management application

Test Case tables can be created for example with text processing or spreadsheet programs. A test case table consists of test case id, prerequisites, step definition, input, expected outcome and special considerations. Those tables might be quite large, so a good tip is to keep the page orientation in a real document as landscape, not portrait. A test case might consist of various steps and the result of the first step can be used as input for the second step.

Test Case 1 – Insert Patient

Prerequisites:

- Login into the system has to be successfully completed
- Main Dialog is displayed for the user

Test Case 1.1: Correct saving (user has inserted valid data to the text fields)

Step	Input	Expected outcome	Special Considerations
1	Fill all text fields Name = Matt Pitt Personal ID = 120775-765R Address = unknown Symptoms= Broken arm	The text inserted by the user is visible in fields	
2	Press Save button	<i>Would you like to save data</i> dialog (Di2)	
3	Press Yes button in Di2 (fields contain valid data)	<i>You have saved data</i> dialog (Di3)	Check whether patient exists in the database

Test Case 1.2: Cancel the data saving

Step	Input	Expected outcome	Special Considerations
1	Fill all text fields: Name = Sam Personal ID = 041081-5678 Address = Wall Street 73 Symptoms= Head Ache	The text inserted by the user is visible in fields	
2	Press Save button	<i>Would you like to save data</i> dialog (Di2)	
3	Press Cancel button in Di2	Main dialog (Di1)	
4	Press Ok button in Di3	Main dialog (Di1)	Check that data was not saved to the database

Test Case 1.3 Incorrect saving (user has inserted non-valid data to the text fields)

Step	Input	Expected outcome	Special Considerations
1	Fill text fields incorrect or leave a required field empty Name *= empty Personal ID = 150984-543 Address = Symptoms=	The text inserted by the user is visible in fields (with errors)	
2	Press Save button	<i>Would you like to save data</i> dialog (Di2)	
3	Press Yes button	<i>Incorrect input</i> dialog (Di4)	
4	Press Ok button	Main dialog (Di1)	

5 Testing action flow**5.1 Test Cases for Component System**

This example focuses on testing Lab Test Business Component System. In figure 9, there is Lab test business component system. At the process layer there is one business component, Lab test workflow. At the entity layer there are Lab test, Test result analyzer, Department and Patient business components. Utility layer business components include Lab test codebook and Addressbook. Database integrity manager and Performance monitor are auxiliary business components.

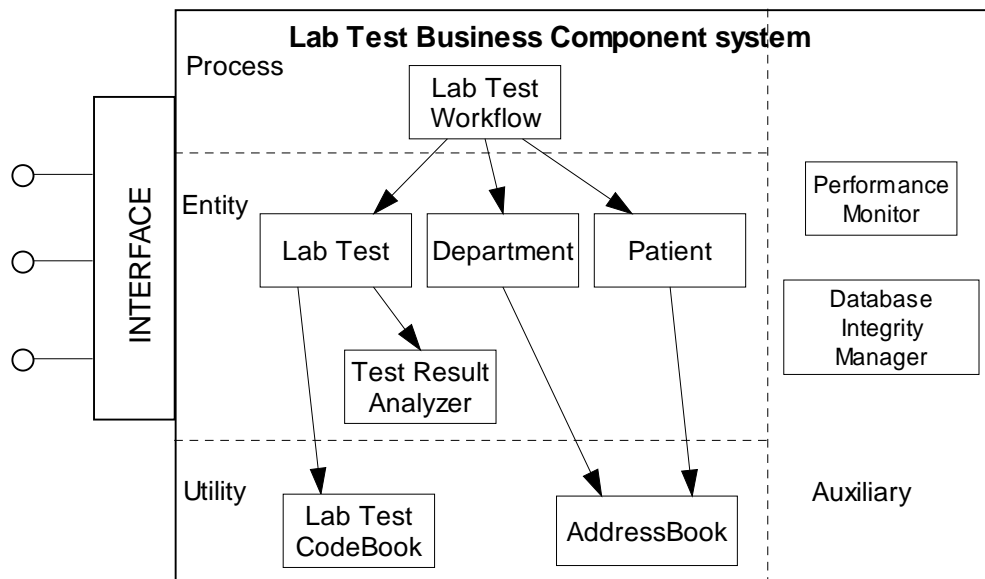


Figure 9. Lab Test BCS

Most important action flows has to be tested. The action flow describes functions between several people and several systems. Action flows are derived from requirement specification. The essential point is that one function follows another in right order. If in our example the patient has not been created in the system, it's not possible to insert medical data about patient to the system.

One action flow of Lab Test BCS is following:

- Create or choose a patient; (human and Patient BC)
- Examine patient
- Create a lab order; (human and Lab Test BC)
- Send the lab order to the lab; (human, Lab Test and Department BC)
- Reception
- Take a sample; (human)
- Analyze the sample and return results; (Test Result Analyzer BC)
- Derive reference values; (Lab Test Codebook BC)
- Save lab test results and reference values; (Lab Test BC)

This action flow is shown as a Use Case diagram in Figure 10.

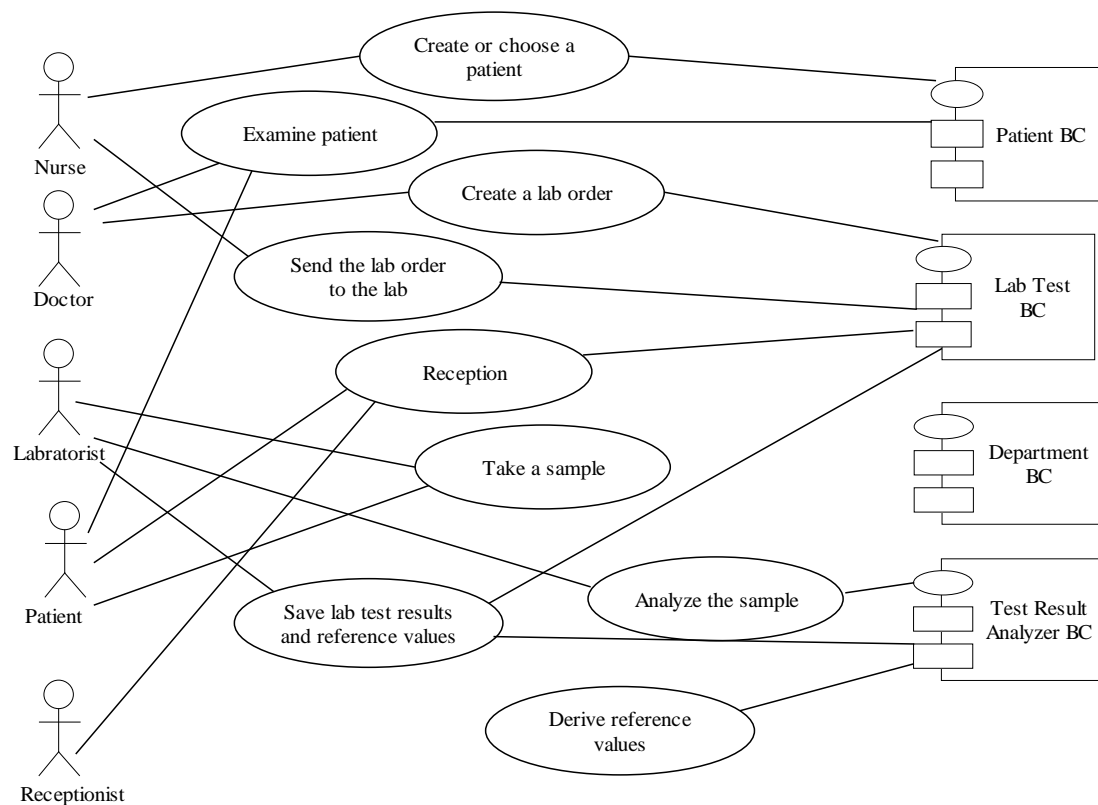


Figure 10: A Use Case diagram for Lab Test BCS

Following table shows a test case designed for this action flow. Test case is based on the action flow and use case diagram of Lab Test BCS. It is possible to utilize scenarios, too.

One test case for Lab Test BCS

Step	Input	Expected outcome	Special Considerations
1 Create patient	Create patient button	The basic information about patient updated	If patient exists, it won't be created
2a Examine patient	Search patient	Patient record on the screen	-
2b Examine patient	Examination data about patient	Patient record updated with examination data	Check that updated data exists in the database
3 Create a lab order	Choose lab tests	Lab order saved	Check that lab tests have been saved
4 Send the lab order to the lab	Reserve time from lab. Send the lab order	Time reserved. Lab order sent	Check that the lab order came to the right place
5 Reception	Choose a patient	Lab test order on the screen	Check the time reservation

Step	Input	Expected outcome	Special Considerations
6 Take a sample	-	-	-
7 Analyze the sample	-	-	Check that the analyser works correctly
8 Derive reference values	-	-	Check that the analyser works correctly
9 Save lab test results	Lab test results + Save button	Lab test results saved	Check that lab test results have been saved

Test cases for other action flows are derived similarly.

5.2 Testing Components

Next we consider as an example an operation sequence of Lab Test BC:

- Input a patient number;(human and user DC)
- Find lab test results with reference values;(enterprise and resource DC)
- Output lab test results with reference values;(user DC)
- Evaluate results;(human)
- Decide further actions;(human)
- Send lab test results and advice for further actions to the department;(human, user and enterprise DC)

The operation sequence is shown as a Use Case Diagram in Figure 11.

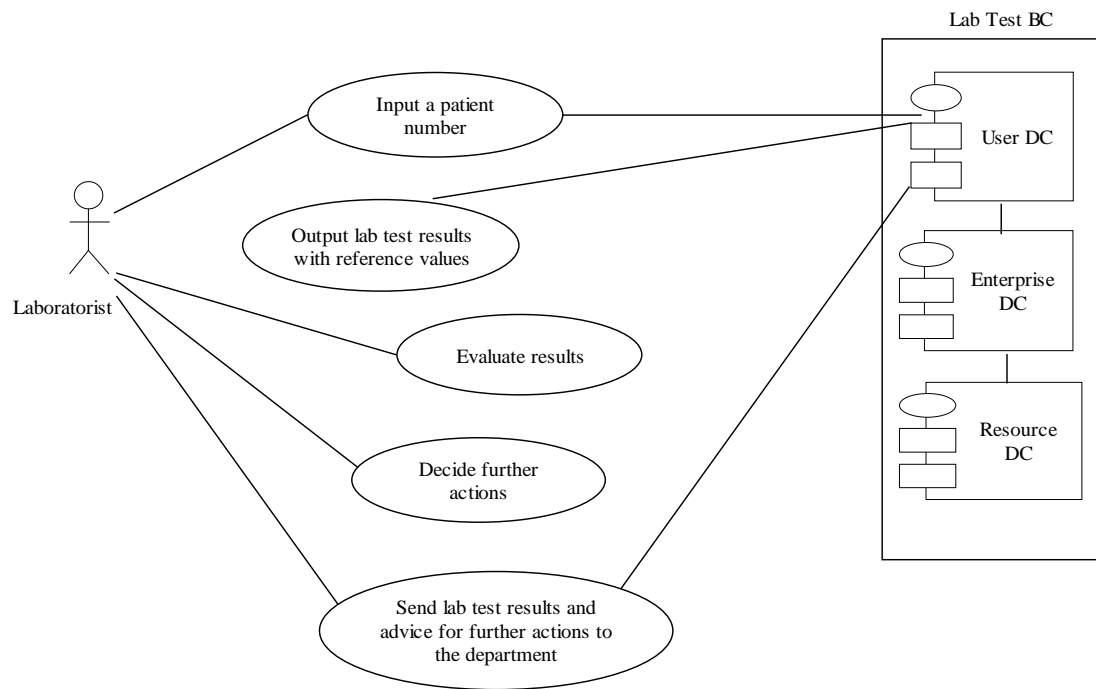


Figure 11: Save lab test results and reference values – a lower level description

This operation sequence can be tested with following test case:

One test case for Lab Test BC

Step	Input	Expected outcome	Special Considerations
1 Input a patient number	Patient number	Patient number has been entered	This step will be done together with step 2
2 Output lab test results...	Press Find button when patient number has been entered.	Lab test results with reference values on the screen	-
3 Evaluate results	-	-	-
4 Decide further actions	-	-	-
5 Send lab test results ...	Lab test results + Send button	Lab test results sent to the department	Check that department received lab test results

At the lowest level we test the interfaces of the distributed component. This can be done in the same way like testing an application with one dialog (See chapter 3).

6 Test Report

A test report is a document that a tester follows while testing applications. At the beginning of the test report there is general information about the project and test cases, for example: project name, test case description and id, test level, test environment, test technique and referred documents, names of the tester and the person who fixes errors.

The test report table includes columns of test case table inserted with actual result, information about whether the test case was passed or failed, possible defect, severity of the defect, the date when the defect was fixed and person who fixed the defect.

Test Case Reports were made for

- Calculate Payment application (pages 15-17)
- Patient Management application
- Lab Test BCS (Similarly)
- Save Lab Test Results (Similarly)

Project:	TEHO	Test Level:	Functional
Test Case:	TC1 Calculate Payment	Environment:	WinNT
Description:	Calculate payment for the patient	Technique:	Equivalence Partitioning
Author:	Marko Jäntti	Referred	
Date:	1.2.2002	Documents:	Requirements Specification
Fixed by:	David Designer (DD)		

Id	Prerequisites	Test Input	Expected result	Actual result	P/F*	Defect and severity**	Fixed (Date)
1.1		Patient type=male Age= -1 Press calculate	Error Message	Ok	P		
1.2		Patient type=male Age= 5 Press calculate	Error Message	Ok	P		
1.3		Patient type=male Age= 18 Press calculate	Payment is 100 euro	Ok	P		
1.4		Patient type=male Age= 36 Press calculate	Payment is 120 euro	Ok	P		
1.5		Patient type=male Age= 51 Press calculate	Payment is 140 euro	Payment 160	F	Error in calculation, S2	DD 13.3
1.6		Patient type=male Age= 146 Press calculate	Error Message	Ok	P		
1.7		Patient type=male Age= @#111 Press calculate	Error Message	Ok	P		

* Pass/Fail ** Severity of defect: S1 = Critical, S2 = Major, S3 =Average, S4 = Minor

Id	Prerequisites	Test Input	Expected result	Actual result	P/F*	Defect and	Fixed (Date)
----	---------------	------------	-----------------	---------------	------	------------	--------------

							severity**	
1.8	Patient type=female Age= -5 Press calculate	Error Message	Ok	P				
1.9	Patient type=female Age= 2 Press calculate	Error Message	No error message	F			Error message must be displayed	DD 12.3
1.10	Patient type=female Age= 18 Press calculate	Payment is 100 euro	Ok	P				
1.11	Patient type=female Age= 31 Press calculate	Payment is 120 euro	Ok	P				
1.12	Patient type=female Age= 51 Press calculate	Payment is 140 euro	Ok	P				
1.13	Patient type=female Age= 146 Press calculate	Error Message	Ok	P				
1.14	Patient type=female Age= aWqi Press calculate	Error Message	Ok	P				

* Pass/Fail ** Severity of defect: S1 = Critical, S2 = Major, S3 =Average, S4 = Minor

Id	Prerequisites	Test Input	Expected result	Actual result	P/F*	Defect and severity**	Fixed (Date)
1.15		Patient type=child Age= 45 Press calculate	Error Message	Ok	P		
1.16		Patient type= child Age= 13 Press calculate	Payment is 50 euro	Ok	P		
1.17		Patient type= child Age= 18 Press calculate	Error Message	Ok	P		
1.18		Patient type= child Age= 42 Press calculate	Error Message	Ok	P		
1.19		Patient type= child Age= 140 Press calculate	Error Message	java.lang.ArithmeticException	F	Exception handling doesn't work	DD 14.3
1.20		Patient type= child Age= 146 Press calculate	Error Message	Ok	P		
1.21		Patient type= child Age= xxciji Press calculate	Error Message	Ok	P		

* Pass/Fail ** Severity of defect: S1 = Critical, S2 = Major, S3 =Average, S4 = Minor

Project:	TEHO	Test Level:	Functional
Test Case:	TC1 Insert Patient	Environment:	WinNT
Description:	Insert patient information to the database TC1.1 Correct saving TC1.2 Cancel saving TC1.3 Incorrect saving	Technique:	Equivalence Partitioning
Author:	Marko Jäntti	Referred Documents:	Requirements Specification 1.0: Activity Diagram
Date:	1.2.2002		

Id	Prerequisites	Test Input	Expected result	Actual result	P/F	Defect and severity	Fixed (Date)
TC1.1 Step1	Main Dialog displayed	Fill all text fields: Name = Sam, Personal ID = 041081-5678, Address = Wall Street 73 Symptoms= Head Ache	The text inserted by the user is visible in fields	Ok	P		
Step2	Text fields contain data	Press Save-button (Di1.B1)	<i>Would you like to save data</i> dialog (Di2)	Ok	P		
Step3	Di2 displayed	Press Cancel button in Di2	<i>Main dialog</i> (Di1)	Ok	P		

* Pass/Fail ** Severity of defect: S1 = Critical, S2 = Major, S3 =Average, S4 = Minor

Id	Prerequisites	Test Input	Expected result	Actual result	P/F*	Defect and severity	Fixed (Date)
TC1.2 Step1	Main Dialog displayed	Fill all text fields: Name = Sam, Personal ID = 041081-5678, Address = unknown Symptoms= Head Ache	The text inserted by the user is visible in fields	Ok	P		
Step2	Fields contain valid data	Press Save-button (Di1.B1)	<i>Would you like to save data</i> dialog (Di2)	Ok	P		
Step3	Di2 displayed	Press Yes button in Di2	You have saved data dialog (Di3)	Ok	P		
Step 4		Press Ok in Di3	Main dialog (Di1)	Di3 still displayed	F	Button listener doesn't work, S3	DD 16.3

Special Considerations: TC 1.2 Step 3 Check whether patient with valid values exists in the database

* Pass/Fail ** Severity of defect: S1 = Critical, S2 = Major, S3 =Average, S4 = Minor

Id	Prerequisites	Test Input	Expected result	Actual result	P/F*	Defect and Severity**	Fixed (Date)
TC1.3 Step1	Main Dialog displayed	Fill text fields with invalid values Name = empty Personal ID = 041081-5678, Address = unknown Symptoms=""	The text inserted by the user is visible in fields (with errors)	Ok	P		
Step2	Fields contain invalid data	Press Save-button (Di1.B1)	<i>Incorrect input dialog</i> Di4	Dr. Watson Fatal error	F	System went down, S1	DD 15.3
Step3	Di4 displayed	Press Ok button in Di4	<i>Main dialog (Di1)</i>	Ok	P		

Special Considerations:

* Pass/Fail ** Severity of defect: S1 = Critical, S2 = Major, S3 =Average, S4 = Minor

Rational Test Tools

Marko Jäntti, Mirja Immonen	Kuopion yliopisto Tietojenkäsittelytieteen ja sovelletun matematiikan laitos
-----------------------------	---

Content

1	INTRODUCTION	3
2	RATIONAL TESTSTUDIO	3
2.1	RATIONAL ADMINISTRATOR.....	4
2.1.1	Goal.....	4
2.1.2	Purpose.....	4
2.1.3	How to use Rational Administrator.....	4
2.1.4	Evaluation	5
2.2	RATIONAL TESTMANAGER.....	6
2.2.1	Goal.....	6
2.2.2	Purpose.....	6
2.2.3	How to use Rational TestManager	6
2.2.4	Evaluation	8
2.3	RATIONAL ROBOT	9
2.3.1	Goal.....	9
2.3.2	Purpose.....	9
2.3.3	How to use Rational Robot	9
2.3.4	Evaluation	10
2.4	RATIONAL PURECOVERAGE	11
2.4.1	Goal.....	11
2.4.2	Purpose.....	11
2.4.3	How to use PureCoverage.....	11
2.4.4	Evaluation	15
2.5	RATIONAL PURIFY.....	15
2.5.1	Goal.....	15
2.5.2	Purpose.....	15
2.5.3	How to use Purify	15
2.5.4	Evaluation	17
2.6	RATIONAL QUANTIFY.....	17
2.6.1	Goal.....	17
2.6.2	Purpose.....	17
2.6.3	How to use Quantify	17
2.6.4	Evaluation	20
2.7	RATIONAL TESTFACTORY	20
2.7.1	Goal.....	20
2.7.2	Purpose.....	20
2.7.3	How to use TestFactory	21
2.7.4	Evaluation	22
3	SUMMARY OF RATIONAL TEST TOOLS	22
4	OTHER TESTING PROGRAMS.....	22
4.1	SILK PRODUCT FAMILY FOR TESTING.....	22

1 Introduction

The purpose of this document is to describe how test programs can be used in software engineering process to help members of the project to manage test activities cost-efficiently.

Software testing should not be anymore a complex and uncontrollable process. Project managers need more detailed information how much money and time test activities will require in order to plan project resources.

2 Rational TestStudio

Rational TestStudio consists of following products:

- Rational Administrator (create a project, manage users and groups)
- Rational TestManager (manage all test activities, organize test plans and cases)
- Rational Robot (create manual and automated scripts)
- Rational PureCoverage (control the coverage of program code)
- Rational Purify (monitor defects and memory usage)
- Rational Quantify (monitor performance of your application)
- Rational TestFactory (create application map, combine automatic test generation with source code coverage analysis)

2.1 Rational Administrator

2.1.1 Goal

Rational Administrator centralizes the management of a software project in one place and manages relationships between users, groups of users and Rational Applications.

2.1.2 Purpose

An administrator can create new projects and manage user privileges. Project can be connected with

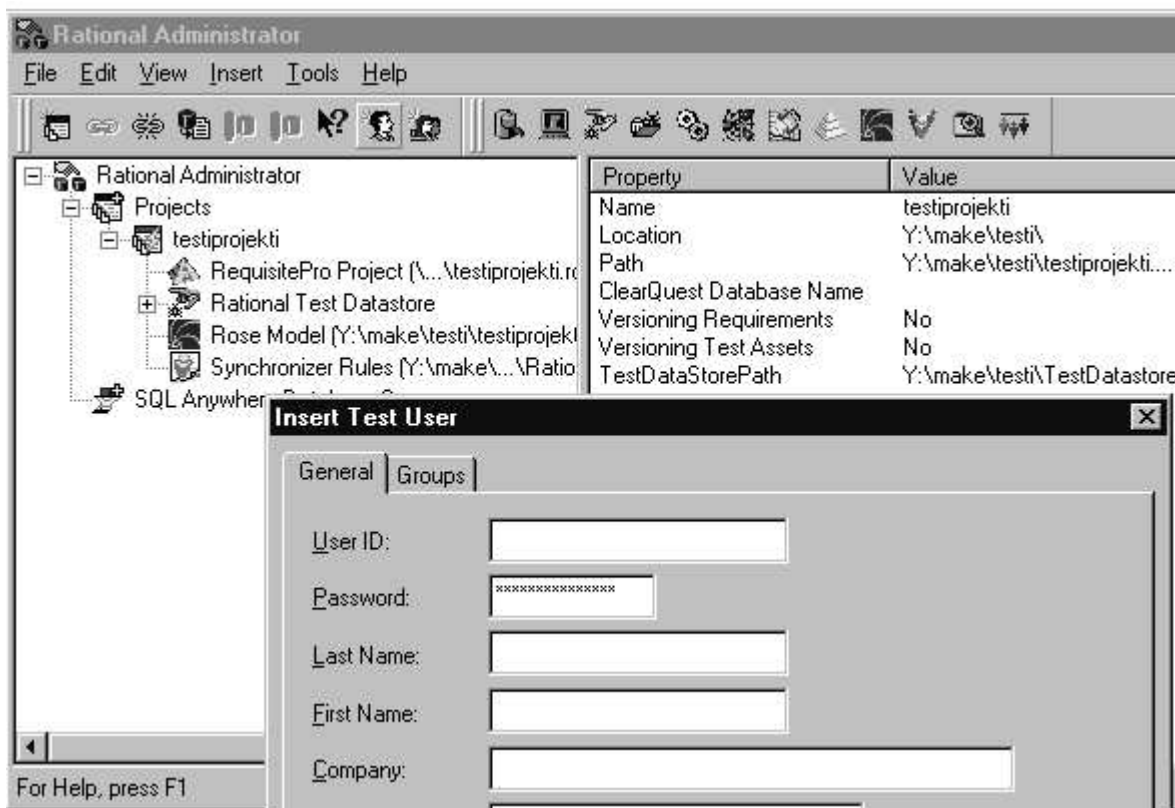
- Rational Test datastore which stores application testing information such as test plans, test cases, logs, reports and builds.
- Rational RequisitePro project which stores requirements documents and a dynamically-linked database.
- Rational ClearQuest database which stores change-request information such as defect reports and documentation modifications
- Rational Rose models that stores visual models for business processes, software components, classes and objects and distribution and deployment processes.

2.1.3 How to use Rational Administrator

You can use Rational Administrator to

- create a new project with or without change management
- configure project
- register project
- delete project
- create and manage users and groups

Rational Administrator/Insert test user



2.1.4 Evaluation

Rational Administrator is a tool for a project manager of the test project. Administrator must be used together with other Rational software products.

Rational Administrator is quite easy to use for advanced users but we noticed some difficulties while we used the program:

- You are not able to insert test users before you connect the project (File/Connect and then enter password)
- You are not able to use all functions of TestManager or some other programs if you have insufficient privileges (to get them, connect the project in Administrator, insert test group with required privileges and ensure that a test user becomes part of the inserted test group).

2.2 Rational TestManager

2.2.1 Goal

Rational TestManager is a tool for managing all test activities, test assets and data. TestManager can be used in planning, implementing and evaluating tests.

2.2.2 Purpose

Software developers, testers and business analysts can see test results from their own view and are able to use the information in their own work:

Analysts know whether business requirements have been tested and do they work. Software developers can see test results of their own program code and testers can plan the test coverage in relation to test plan. Software developer team can easier focus on quality assurance.

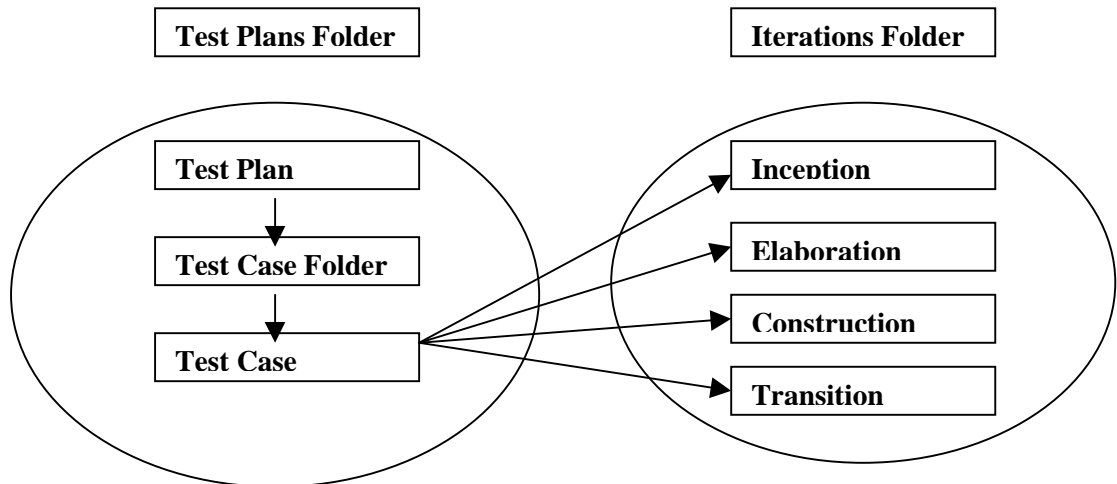
2.2.3 How to use Rational TestManager

When you use TestManager, you can

- Manage all test activities and test material in one application platform
- Track team work to quality requirements
- Organize all test artifacts to one test plan
- Evaluate the execution of tests
- Analyse test coverage through on-line reports

A test user can create a new test plan with test case folders that include test cases. It's possible to associate a test case with created iterations that are phases of software engineering process. A test user can also rename iteration. The advantage of organizing test cases is an easier way to find out which tests should be executed in each phase.

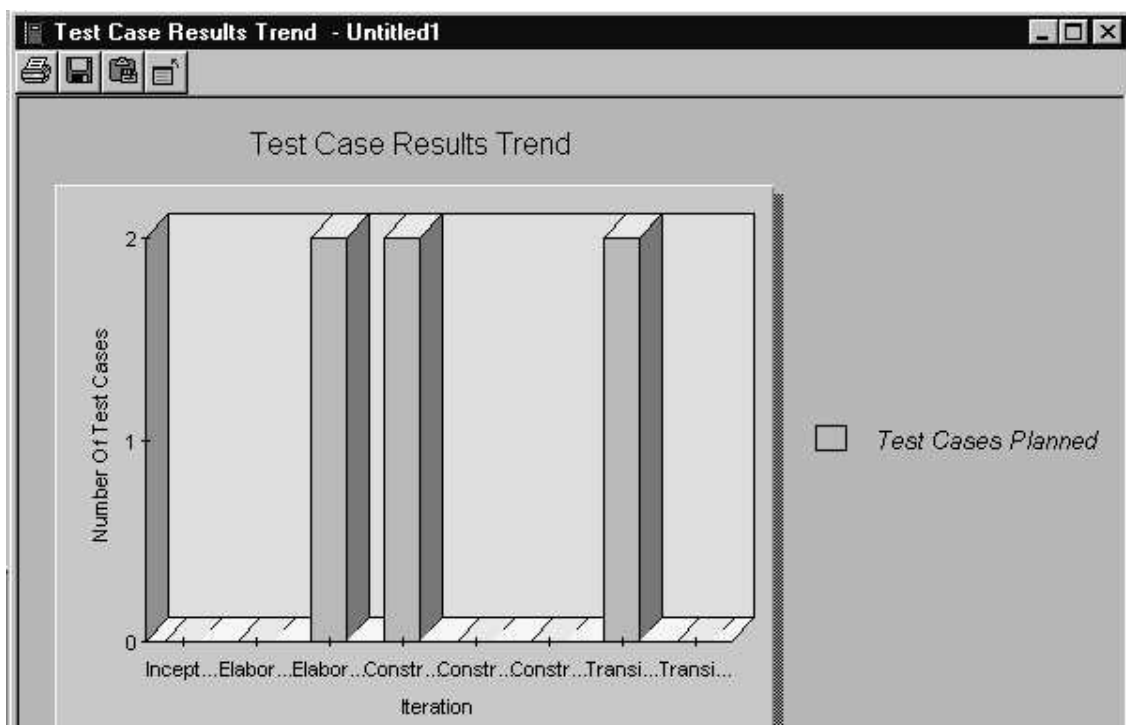
Test plans and iterations



Test case inputs can be associated with requirements or use cases created in RequisitePro-project (requirements specification). Test case properties can include links to external documents, for example use case documentation or specification documentation. Test case implementation can consist of either manual or automatically created scripts.

Test user is able to analyse test coverage with on-line reports, for example test case distribution or test case results trend as shown in the following picture.

On-line report of test case results



2.2.4 Evaluation

TestManager program has an important role in Rational TestStudio because it manages all test activities and collects test information about other programs in TestStudio.

Creating test plans, test cases and iterations is quite simple to do. In the same way, there are no difficulties in creating on-line reports and associating test cases with iterations, external documents and requirements in RequisitePro project.

However, running test cases seems to be more complex. When a test user runs a test case, program displays many forms the purpose of which is not clear enough. It is not so simple to mark test cases failed and passed.

2.3 Rational Robot

2.3.1 Goal

Rational Robot is a tool for developing scripts for functional testing and performance testing. Robot helps you to plan, develop, execute and analyze tests.

2.3.2 Purpose

Robot scripts can be GUI scripts (for functional testing) or VU and VB scripts (for performance testing).

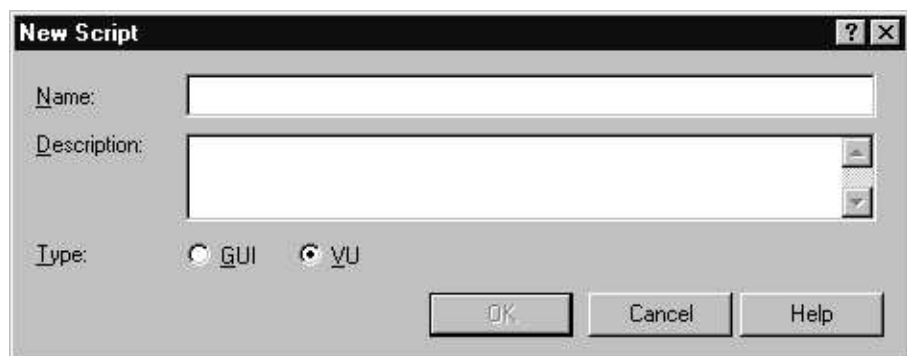
Rational Robot can be used to:

- Record and play back scripts which go through the application
- Create and edit scripts with SQABasic scripting (Rational's SQABasic language based on Microsoft's Visual Basic), VU (based on the C programming language) and VB scripting
- Collect application information with diagnostic tools (Purify, Quantify, PureCoverage) while script is being played.

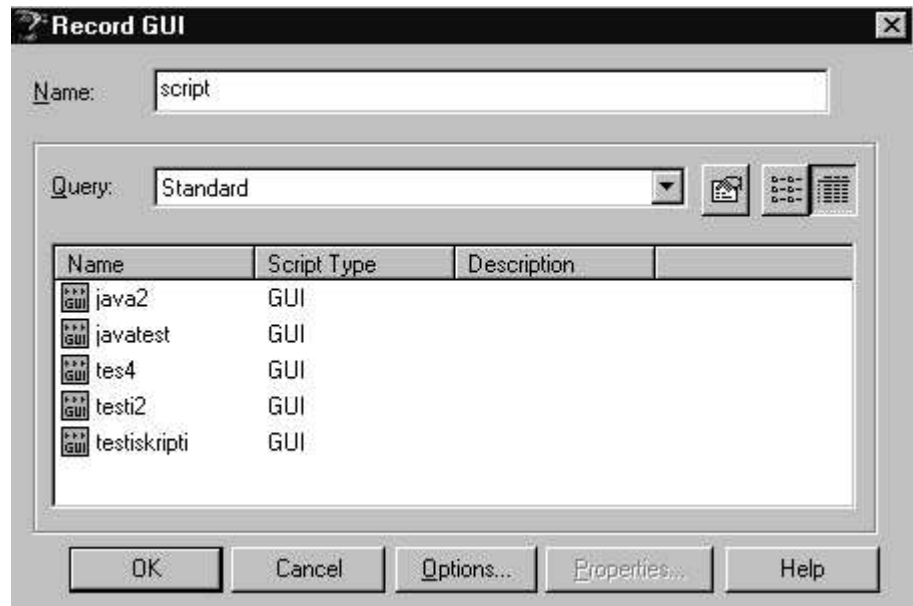
2.3.3 How to use Rational Robot

First create a script and give the name for your script. The type of the script is GUI (record graphical user interface activities) or VU (collects client/server requests).

Creating a new script in Robot



Record a new script or update an existing script by choosing *Record* command.



Robot records activities (mouse moves and key enters) when the following dialog is displayed for the user:



2.3.4 Evaluation

Rational Robot is quite logical to use for GUI recording but there could be more instructions how to start recording VU scripts (What should be entered to the *Start Application* dialog).

A test user must create a new script if somebody makes changes to the graphical user interface, for example the test user creates Script 1 for

User Interface 1. A system developer changes places of two buttons in UI 1 -> creates UI 2.

Script 1 doesn't work anymore because coordinates of two buttons are different in UI 1 and UI 2. Script 1 tries to find buttons in coordination points but buttons aren't there. A good feature in test software would be if the test user didn't have to create Script 2 for new user interface. A software tester has to keep in his mind that scripts don't automatically know whether the actual result (for example the calculation of weight index) is right or wrong.

Example test script:

```
InputKeys "{~}mimmonen/Painoindeksi.html{ENTER}"  
    JavaWindow Click, "ObjectIndex=1", "Coords=179,20"  
    InputKeys "1.80{TAB}75"  
    JavaWindow Click, "ObjectIndex=1", "Coords=232,54"
```

2.4 Rational PureCoverage

2.4.1 Goal

Rational PureCoverage collects detailed information about code coverage for test application and its components.

2.4.2 Purpose

During the software engineering process software will change daily or even more often and testing activities might be late. PureCoverage helps to check program code and informs whether each line, method or function of the program code is used while running the application.

2.4.3 How to use PureCoverage

PureCoverage program helps you to

- Collect code coverage information
- Analyze code coverage information
- Run program and tests

With PureCoverage you can see how many calls coverage items (classes or methods) receive and how many methods or lines are missed or hit while program is running. PureCoverage can be used for

- Visual C/C++ code in .exes, .dlls, OLE/ActiveX controls, and COM objects
- Visual Basic projects, p-code, and native-compiled code in .exes, .dlls, OLE/ActiveX controls, and COM objects
- Java applets, class files, JAR files, and code launched by container programs in conjunction with a Java virtual machine (JVM)
- Managed code in any source language that has been compiled for Microsoft's .NET platform, including C#, Visual C/C++, and Visual Basic, in .exes, .dlls, OLE/ActiveX controls, and COM objects
- Components launched by container programs
- Microsoft Excel and Microsoft Word plug-ins

PureCoverage creates a list of all modules and functions in the application

Coverage Browser: java.exe							
Module View File View							
Coverage Item	Calls	Methods Missed	Methods Hit	% Methods Hit	Lines Missed	Lines Hit	% Lines Hit
Run @ 03.12.2001 08:51:44 -classic Ohjelma	174	58	49	45,79	367	220	37,9
Esine	78	9	15	62,50	26	53	67,3
(Unknown Directory)	78	9	15	62,50	26	53	67,3
Esine.java	78	9	15	62,50	26	53	67,3
Esine.<init>()	11		hit		0	1	100,0
Esine.<init>(int,int,int,java.lang.String,java.lang.String)	1		hit		0	9	100,0
Esine.asetaAjankohta(java.lang.String)	1		hit		0	2	100,0
Esine.asetaEkaomistajanumero(int)	1		hit		0	2	100,0
Esine.asetaEsinenumero(int)	1		hit		0	2	100,0
Esine.asetaKolmasomistajanumero(int)	1		hit		0	2	100,0
Esine.asetaKuvaus(java.lang.String)	1		hit		0	2	100,0
Esine.asetaNimi(java.lang.String)	1		hit		0	2	100,0
Esine.asetaTokomistajanumero(int)	1		hit		0	2	100,0
Esine.asetaValmistusvuosi(int)	1		hit		0	2	100,0
Esine.jarjestys(Esine)	0	missed			3	0	0,0
Esine.kirjoitaTiedostoon(java.io.RandomAccessFile)	0	missed			9	0	0,0
Esine.kirjoitaMerkitaulu(java.io.RandomAccessFile)	0	missed			3	0	0,0
Esine.lueTiedostosta(java.io.RandomAccessFile)	11		hit		0	10	100,0
Esine.lueMerkitaulu(java.io.RandomAccessFile, char)	33		hit		0	3	100,0
Esine.nimi()	0	missed			1	0	0,0
Esine.oNumero1()	0	missed			1	0	0,0
Esine.oNumero2()	0	missed			1	0	0,0
Esine.oNumero3()	0	missed			1	0	0,0
Esine.sallittuPituus(java.lang.String,int)	3		hit		1	5	83,3

PureCoverage compares summaries of application runs

Run Summary: java.exe (Auto Merge)	
Details	
Host Name:	PARTA23
Machine Type:	Intel Pentium Pro Model 6 Stepping 0
# Processors:	1
O/S Version:	Windows NT 4.0.1381 Service Pack 6
Dataset Size (bytes):	69632
(Run 1) PID:	0xd6
(Run 1) Program:	c:\jdk1.3.1\bin\java.exe
(Run 1) Arguments:	-classic Ohjelma
(Run 1) Working Directory:	R:\Teho\java
(Run 1) Start Time:	10.12.2001 12:05:10 [Run]
(Run 1) Stop Time:	10.12.2001 12:05:32
(Run 1) Elapsed Time:	19778 ms
(Run 1) Elapsed Time:	00:00:22
(Run 2) PID:	0xda
(Run 2) Program:	c:\jdk1.3.1\bin\java.exe
(Run 2) Arguments:	-classic Ohjelma
(Run 2) Working Directory:	R:\Teho\java
(Run 2) Start Time:	10.12.2001 12:08:39 [Run]
(Run 2) Stop Time:	10.12.2001 12:09:12
(Run 2) Elapsed Time:	33157 ms
(Run 2) Elapsed Time:	00:00:33

PureCoverage marks code lines with different colours

Methods: Ohjelma.kyselyEsineesta(omistajaRekisteri,e) Colors: ▼		
Line Coverage	Line Number	
	101	* käyttäjä antaa numeron.
	102	* @return luetun omistajan
	103	*/
	104	private static int kysyOm
2	105	int oNumero = 0;
2	106	boolean ok = false;
	107	do {
	108	try {
2	109	Kirjoita.println("Anna omistajanumero.");
2	110	oNumero = Integer.parseInt(in.readLine());
2	+	
2	+	
2	+	
2	111	ok = true;
	112	} catch (NumberFormatException e) {
0	113	Kirjoita.println("Omistajanumeroon vain numeroita kiitos!");
	114	}
2	115	} while (!ok);
2	116	return oNumero;
	117	}

- Hit Lines...
- Missed Lines...
- Dead Lines...
- Summaries...
- Partially Hit Multi-block Lines...
- Use Default Colors

2.4.4 Evaluation

Rational PureCoverage is useful in white box testing. It's important to test every line of the program code (all structures like *while* or *if*). PureCoverage helps developers to see which lines of program code have been tested and which have been missed. They can also check how many times each method is called during the test run.

2.5 Rational Purify

2.5.1 Goal

Runtime errors and memory leaks cause lots of problems for applications because they spend time and it's hard to find them. Rational Purify helps to find and delete them even without the program code.

2.5.2 Purpose

The purpose of Purify is to produce clean and reliable solutions in less time. Purify saves money and time spent for developing software and increases quality.

2.5.3 How to use Purify

Visual C/C++ programmers and testers can use Purify

- With Microsoft Visual Studio: Purify integrates with IDE (Information Definition Environment) and debugger enabling checking run time errors
- Standalone: GUI interface provides tools for pointing errors
- Command line: automated tests check functionality of modules and show dependencies and collisions of the code

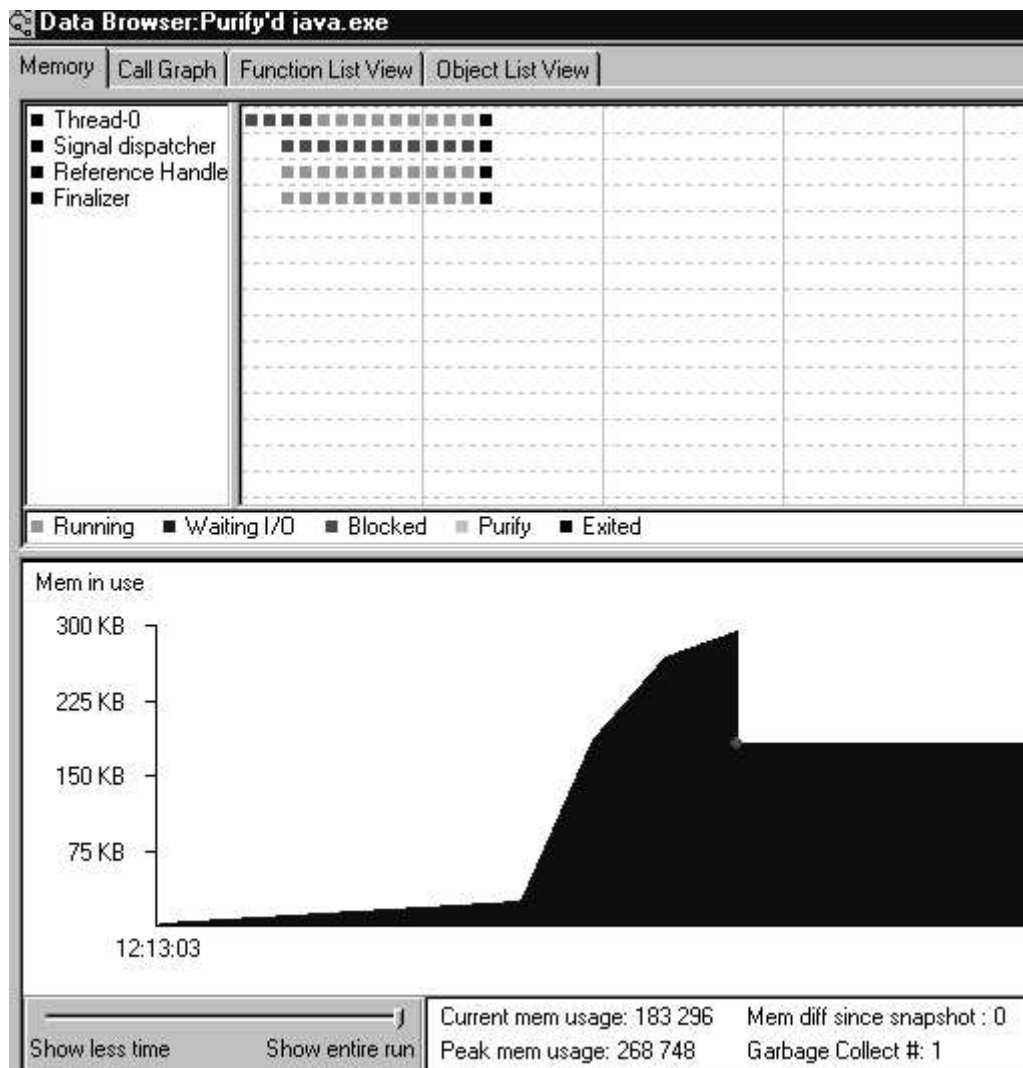
Java programmers and testers can use Purify with Java Virtual Machine to develop and tune the memory efficiency of applets and applications. Purify provides tools for managing memory and

analysing data. Program finds out problems like methods that use too much memory or objects that cause difficulties for garbage collecting. Purify can run applets, Class files, JAR-files and supports container programs like JVM viewer or Internet Explorer.

Purify automatically displays the memory tab when a test user has run a Java program. Memory allocation graph is a graphical representation of the total amount of memory allocated to the program (while the program was running). In addition, Purify creates

- Call graph
- Function list
- Object list

Purify creates a Memory allocation graph



2.5.4 Evaluation

Purify is suitable for describing how much memory the application under test requires during the running process. It would be useful if Purify displayed directly errors or exceptions occurred during the test run. Call graphs are practical tool for programmers or software developers. Function list is rather unclear because it includes all methods (also inherited methods from upper classes).

2.6 Rational Quantify

2.6.1 Goal

Rational Quantify pinpoints performance bottlenecks in Java, Visual Basic and Visual C++ programs.

2.6.2 Purpose

Software developers are able to check with Quantify where their application spends most of its time. Quantify helps you to

- Collect performance data
- Analyze performance data
- Compare performance data

2.6.3 How to use Quantify

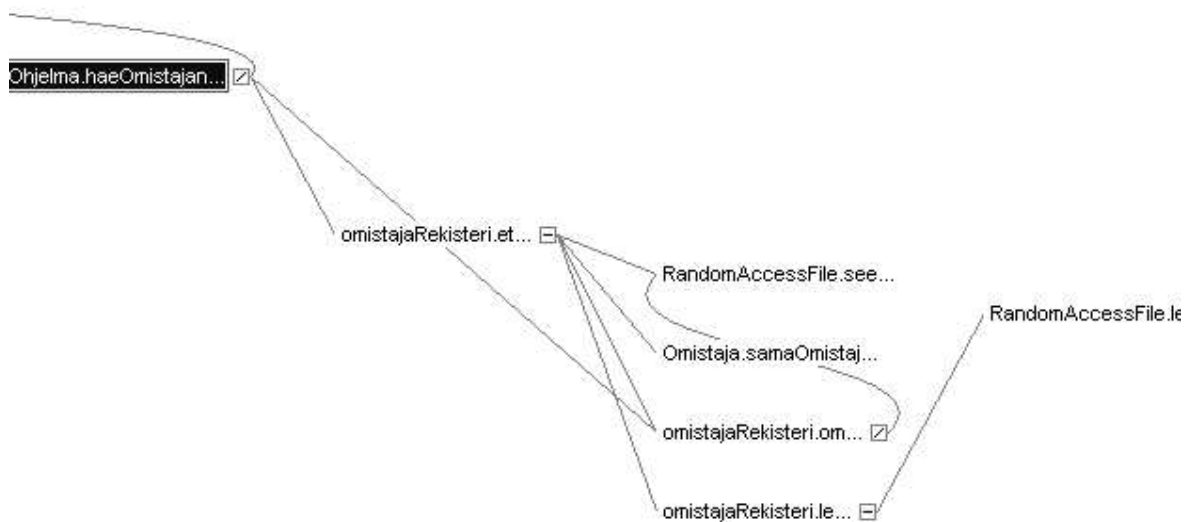
Quantify can measure performance for:

- Visual C/C++ code in .exes, .dlls, OLE/ActiveX controls, and COM objects
- Visual Basic projects, p-code, and native-compiled code in .exes, .dlls, OLE/ActiveX controls, and COM objects

- Java applets, class files, JAR files, and code launched by container programs in conjunction with a Java virtual machine (JVM)
- Managed code in any source language that has been compiled for Microsoft's .NET platform, including C#, Visual C/C++, and Visual Basic, in .exes, .dlls, OLE/ActiveX controls, and COM objects
- Components launched by container programs
- Microsoft Excel and Microsoft Word plug-ins

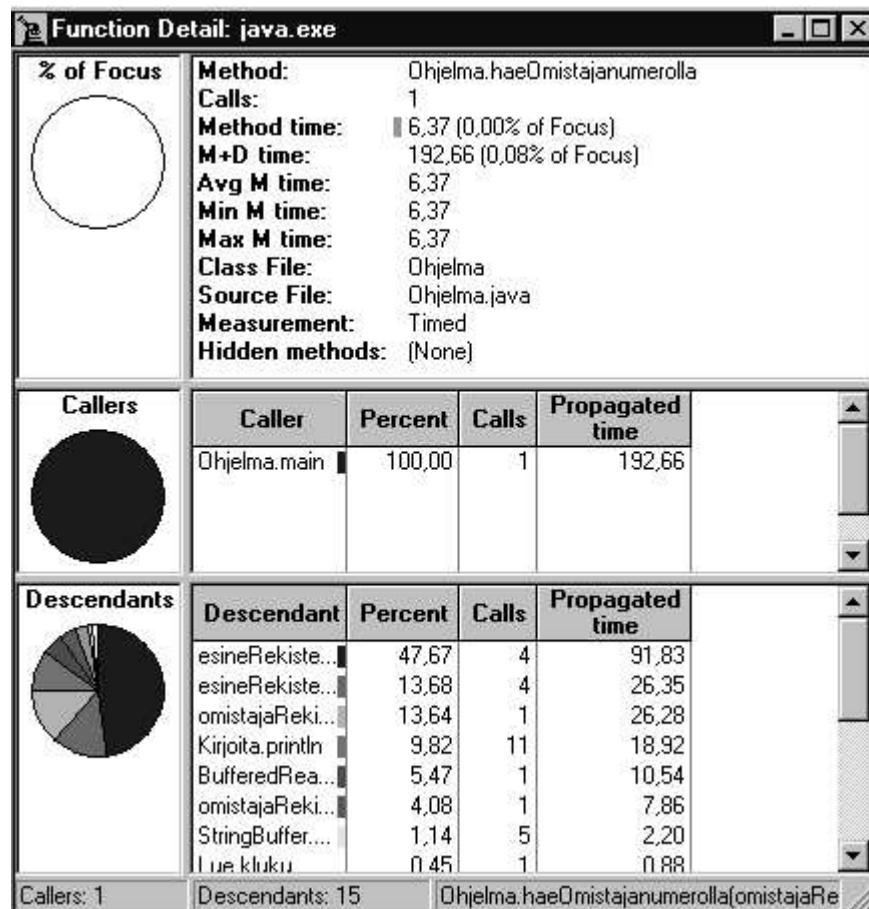
At first Quantify runs your program and creates a call graph of methods in the application.

A call graph of a Java application

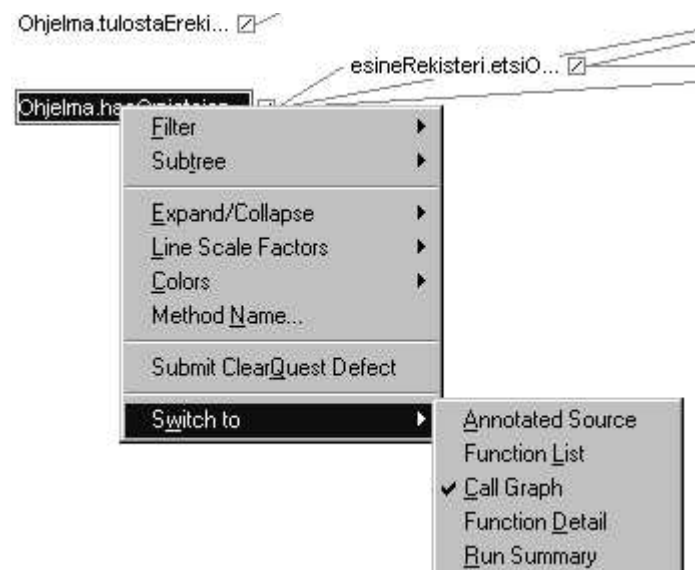


By double-clicking a method of the call graph Quantify displays function details.

Quantify displays function details of the test application



By clicking the right mouse button in the call graph and *Switch to / Annotated Source* you will see how much each line of the program code requires time.



Quantify shows how much time each code line or method spends.

Line time	L+D time	Percent of Method time	Percent of M+D time	Line number	Source
0,13	0,13	0,45	0,10	151	*/
				152	private void kirjoitaMerkkitaulu(RandomAccessFile eTiedo
				153	throws IOException {
				154	*****
					Method: Esine.kirjoitaMerkkitaulu(java.i
					Called: 66 times
					Method time: 29,42 msec (0,08% of Foc
					M+D time: 125,94 msec (0,33% of Foc
					Distribution to Callers:
					Called 66 times. Esine.kirjoitaEtiedoston(java.io.Ran

14,64	14,64	49,77	11,63	155	for (int i=0; i<taulu.length; ++i)
14,64	11...	49,78	88,27	156	eTiedosto.writeChar(taulu[i]);
				157	}

2.6.4 Evaluation

Quantify is quite easy to use for measuring performance of test applications. Program is suitable for software developers in performance testing.

2.7 Rational TestFactory

2.7.1 Goal

Rational TestFactory together with Robot can be used for testing the controls in applications that are created in integrated development environments (IDEs) like Java, Visual Basic and C++.

2.7.2 Purpose

With TestFactory user can

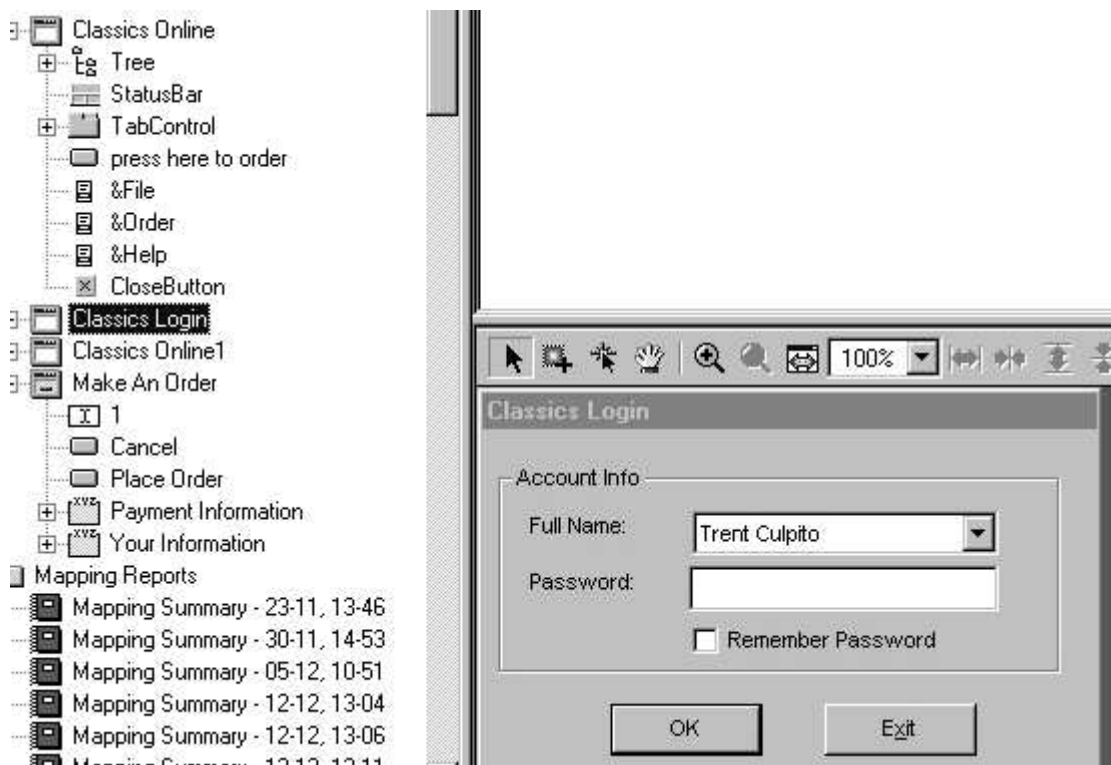
- create and maintain a detailed map of the application-under-test (AUT).
- generate both scripts that provide extensive product coverage and scripts that encounter defects.
- track executed and unexecuted source code and report its detailed findings.

- shorten the product testing cycle by minimizing the time invested in writing navigation code.

2.7.3 How to use TestFactory

The Application mapper creates the application map (explores the user interface of the application-under-test (AUT)). Software developers can check which components are included in the application and what do components look like.

TestFactory displays components in the user interface



2.7.4 Evaluation

This test tool would need more instructions for users. The only thing that our test team could do with TestFactory was creating the component list and program displayed the component like in the user interface.

3 Summary of Rational test tools

The following table shows which programming languages Rational test products will support.

Rational Robot	GUI and client server applications.
Rational PureCoverage	Visual C/C++, Visual Basic, Java, Microsoft Excel and Microsoft Word Plug- ins
Rational Purify	Visual C/C++, Java
Rational Quantify	Visual C/C++, Visual Basic, Java, Microsoft Excel and Microsoft Word plug ins.
TestFactory	Visual Basic, C++ and Java

4 Other testing programs

4.1 Silk product family for testing

SilkTest is a tool for functional testing (Web, Java or client/server applications):

- It includes planning and managing of tests and direct access to find out the content of a database.
- It uses object based 4Test language and has a system that automatically recovers from errors (Testing can be done during night time).
- Test scripts can be reused for various platforms, browsers and other technologies.

SilkPerformer helps system testers in load tests and monitoring. The program for load testing has to create realistic load for application to show its functionality and reliability. Load test includes scalability, realsticity and integration of architectures. Server Analysis Module acts as a part of SilkPerformer and measures efficiency of servers and machines in programming environment.

SilkPilot (Silk for CORBA and EJB) is a tool for CORBA's and EJB/J2EE server's functional and regression testing. SilkPilot can be used for

- automating tests of application server and test behaviour of decentralized objects.
- CORBA server's unit testing through the standard IDL-interface.
- EJB/J2EE server's testing

SilkRadar is a program for defect management to control and manage change requests.

SOFTWARE TESTING TOOLS

Pentti Pohjolainen
Department of Computer
Science and Applied
Mathematics
University of Kuopio
March 2002

CONTENTS

<u>1 Introduction</u>	5
<u>2 References</u>	10
<u>3 Division of the Tools</u>	11
<u>3.1 Test Design Tools</u>	11
<u>3.1.1 Test Case Tools</u>	11
<u>3.1.2 Database Tools</u>	12
<u>3.1.3 Data Generators</u>	13
<u>3.1.4 General Test Design</u>	14
<u>3.2 GUI Test Drivers</u>	16
<u>3.3 Load and Performance Testing Tools</u>	25
<u>3.4 Test Management Tools</u>	32
<u>3.4.1 CORBA</u>	32
<u>3.4.2 C/C++</u>	33
<u>3.4.3 Others</u>	36
<u>3.5 Test Implementation Tools</u>	48
<u>3.5.1 Java</u>	48
<u>3.5.2 C/C++</u>	51
<u>3.5.3 Others</u>	53
<u>3.6 Test Evaluation Tools</u>	58
<u>3.6.1 Java</u>	58
<u>3.6.2 C/C++</u>	60
<u>3.6.3 Others</u>	65
<u>3.7 Static Analysis Tools</u>	69
<u>3.7.1 Java</u>	69
<u>3.7.2 C/C++</u>	71
<u>3.7.3 Others</u>	75

1 Introduction

This work started from the subject of my pro gradu thesis “The newest methods and tools for software testing”. After a long search there were nearly 600 (six hundred) tools found. I am sure, that there are tools much more than these now occurred. A very good aid to me was the list in Internet (www.testingfaqs.org), which Brian Marick made famous and is now maintained by Danny Faught. Other sources have been Internet overall, the brochures of the products and the literature. Because the amount of the tools was so large, I had to restrict them and to take up only the most interesting once. The division to the main groups was: Design, GUI (Graphical User Interface), Load and Performance, Management, Implementation, Evaluation, Static Analysis and outside of inspection: Defect Tracking, Web Sites and Miscellaneous.

The limits between the groups are out of focus, because there are many tools, which can belong to several classes (See Figure 1.).

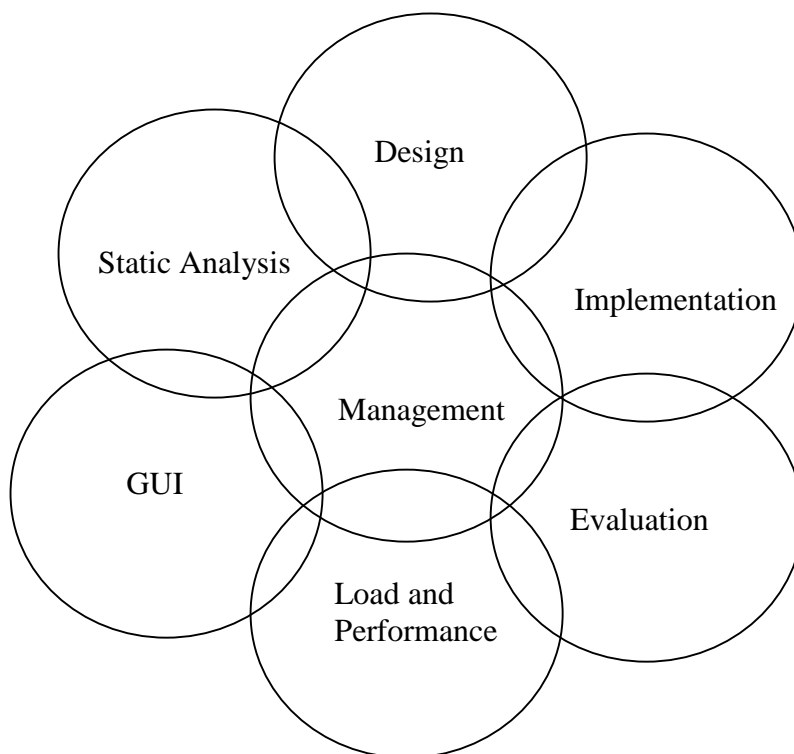


Figure 1. The Division of the tools

A short description of the main groups:

Test Design Tools

Tools that help you decide what tests need to be executed. Test data and test case generators.

Total 15 tools.

GUI Test Drivers

Tools that automate execution of tests for products with graphical user interfaces. Client/server test automation tools, including load testers, also go here.

Total 28 tools.

Load and Performance Tools

Tools that specialize in putting a heavy load on systems (especially client-server systems). These tools are often also GUI test drivers.

Total 21 tools.

Test Management Tools

Tools that automate execution of tests for products without graphical user interfaces. Also tools that help you work with large test suites.

Total 43 tools.

Test Implementation Tools

Miscellaneous tools that help you implement tests. For example, tools that automatically generate stub routines go here, as do tools that attempt to make failures more obvious (assertion generators, etc.)

Total 27 tools.

Test Evaluation Tools

Tools that help you evaluate the quality of your tests. Code coverage tools go here.

Total 31 tools.

Static Analysis Tools

Tools that analyse programs without running them. Metrics tools fall in this category.

Total 33 tools.

Total sum 198 tools.

In PlugIT-project interesting tools are at least: Regression, requirement, component, integration, object-oriented, coverage, test case and use case.

Searching with these words the recent 198 tools the results were:

Regression Tools

Regression testing tools are used to test software after modification. Dividing in groups as above (one or two typical examples per group are presented) there were:

Design: None

GUI: Auto Tester for Windows (No 3) is specifically meant to support project teams in automating regression testing. Others 4, 6, 12, 15 and 27.

Load and Performance: Teleprocessing Network Simulator (No20) can be used to automate regression testing. Others10 and 12.

Management: Test Manager (No 2) provides an interactive development environment for working with regression test suites. OTF – On Object Testing Framework (No 18) is a tool for Smalltalk objects, in which regression testing is automatic with full logging of results. Others 1, 4, 5, 10, 14, 16, 27, 28, 29, 37 and 38.

Implementation: Junit (No 5) is a regression testing framework used by developers who implement unit tests in Java. Others 1, 15 and 18.

Evaluation: Logiscope (No 26) identifies effective non regression if program files have been modified.

Statistic Analysis: ParaSoft Jtest (No 6) automatically performs regression testing of Java code.

Total 28 tools.

Requirement Tools

Requirement-based or requirement definition related tools.

Design: Caliber-RBT (No 1) is a test case design system for requirement-based testing. Others 6 and 15.

GUI: Panorama-2 (No 8) is a tool for requirement analysis. Another 17.

Load and Performance: SilkRealizer (No 19) is a tool that enables users develop and deploy system level testing simulating real world events to assure that applications will meet the requirements.

Management: AutoAdviser (No13) provides from requirements through production a central repository for organizing and managing business requirements, tests and associated files. Others 18, 21, 35 and 37.

Implementation: None

Evaluation: Panorama C/C++ (No 15) is a requirement and test case analysis tool.

Static Analysis: None

Total 12 tools.

Component Tools

Tools, which have some relationships with component-programming.

Design: Inferno's (No 2) capabilities include creation of an intuitive library of reusable components that support shared-scripts and data-driven-scripts.

GUI: None

Load and Performance: None

Management: SilkPilot (No 1) lets you test the behaviour of distributed objects within your application's server components. AutoAdviser (No 13) consolidates your test library components and provides test team members with access to those components. Others 2, 32, 40 and 42.

Implementation: AssertMate for Java (No 2) is a system that aids Java engineers use assertions to ensure proper implementation of component interfaces. Another 1.

Evaluation: QC/Coverage (No 16) helps users by identifying code components that have not been adequately tested.

Static Analysis: WhiteBox (No 33) provide insight into the complexity of different components of the software.

Total 11 tools.

Integration Tools

Tools used with integration testing.

Design: ObjectPlanner (No 13) allows software developers and managers to calculate the approximate time schedules to perform unit and integration testing. Another 15.

GUI: Solution Evaluation Tool (No 14) is usable in testing the integration of new applications.

Load and Performance: None

Management: Cantata (No 3) is a solution for unit and integration testing of C and C++ code. Others 5, 6, 14, 25, 32 and 40.

Implementation: Autotest (No 14) is an automatic test harness tool for integration testing. Another 15.

Evaluation: Cantata++ (No 7) is an effective tool for C++ integration testing. Others 18 and 27.

Static Analysis: None.

Total 15 tools.

Object-oriented Tools

Tools used specially with object-oriented systems. All Java and C++ tools fall automatically in this category although the search has not found them with keyword object.

Design: T-VEC Test Generation System (No 15) is integrated in an environment to support the development and management of structured or object-oriented requirements specifications. Another 12.

GUI: Vermont HighTest Plus (No 23) includes object-level record/playback of all Windows and VB controls. Others 4, 8, 16, 17, 18 and 27.

Load and Performance: None

Management: TOOTSIE (No 42) is a total object-oriented testing support environment. Others 1, 2 and 18.

Implementation: ObjectTester (No 11) is a software tool to automate the generation of C++ unit test scripts. Others 1 and 2.

Evaluation: TotalMetric for Java (No 1) is a software metrics tool to calculate and display object-oriented metrics for the Java language.

Static Analysis: ObjectDetail (No 12) is a software tool to help automate the metrics generation of C++ programs. Another 27.

Total 19 tools.

Coverage Tools

Code coverage, test case coverage, test coverage and so on.

Design: Caliber-RBT (No 1) uses the requirements as a basis to design the minimum number of test cases needed for full functional coverage. Others 2, 4 and 15.

GUI: Smalltalk Test Mentor (No 15) automatically gathers execution and method coverage metrics. Others 8, 18 and 20.

Load and Performance: DataShark (No 6) generates the minimal number of test cases with maximum coverage based on equivalence class partitioning and boundary condition analysis.

Management: Cantata (No 3) is a code coverage analyser. Others 10, 13 and 38.

Implementation: None

Evaluation: DeepCover (No 2) provides test coverage analysis for C/C++ and Java applications. ObjectCoverage (No 13) a branch coverage analysis tool for C++. Others all but 1, 21 and 24.

Static Analysis: Plum Hall (No 15) is a static analysis and code coverage tool for testing C/C++ code. Others 21, 31 and 33.

Total 45 tools.

Test case Tools

Tools used e.g. in test case design.

Design: Validator/Req (No 6) performs test case generation. Others 1, 2, 3, 4 and 5.

GUI: imbus GUI Test Case Library (No 26) is a library of predefined test cases for automated GUI testing. Others 8 and 19.

Load and Performance: DataShark (No 6) generates the minimal number of test cases with maximum coverage based on equivalence class partitioning and boundary condition analysis.

Management: QCIT (nr 21) tracks the software testing process from requirement development, through test plan and test case development and execution. Test Case Manager-TCM (No 30) organizes test cases for storage and execution logging. Others 5, 10, 25, 26, 28, 31, 34, 35 and 40.

Implementation: Autotest (No 14) controls a series of test cases on different programs and software units Visula. Others 11 and 13.

Evaluation: Object/Coverage (No 13) analyses statements and generates a test case coverage report. Others 15 and 24.

Static Analysis: Poly Space Verifier (No 16) is a tool designed to directly detect run-time errors and non-deterministic constructs at compilation time. Others 15, 31 and 33.

Total 31 tools

Use case Tools

Use case testing and design.

Every group none.

Total sum 161 tools.

If you are interested for example in regression tools, you have them now in the same group and you don't have to scan through all the 198.

Here you had two examples how to divide tools in the groups. There are many other grounds to do the same thing. Everyone can think himself what is the best.

We can also place the different types of tools in the software development life cycle (Figure 2.). The limits between groups are ambiguous. The division is based on the divisions by Fewster and Graham [FeG99] and Tervonen [Ter00].

Test management tools can be used in the whole software development life cycle. Test design and inspection tools can be used in requirement specification, in architectural design and in the detailed design phases. The static analysis tools help testing in the coding phase. Execution and comparison tools can be used overall on the right side of V-model. Dynamic analysis tools are usable in functionality, integration and unit testing. They assess the system while the software is running. Coverage tools are designed specifically for unit testing. Acceptance and system tests fall in load and performance tools. GUI test drivers have features of many other tools and are useful in the whole implementation and evaluation area, but they are designed for GUI testing and are distinctly an own group.

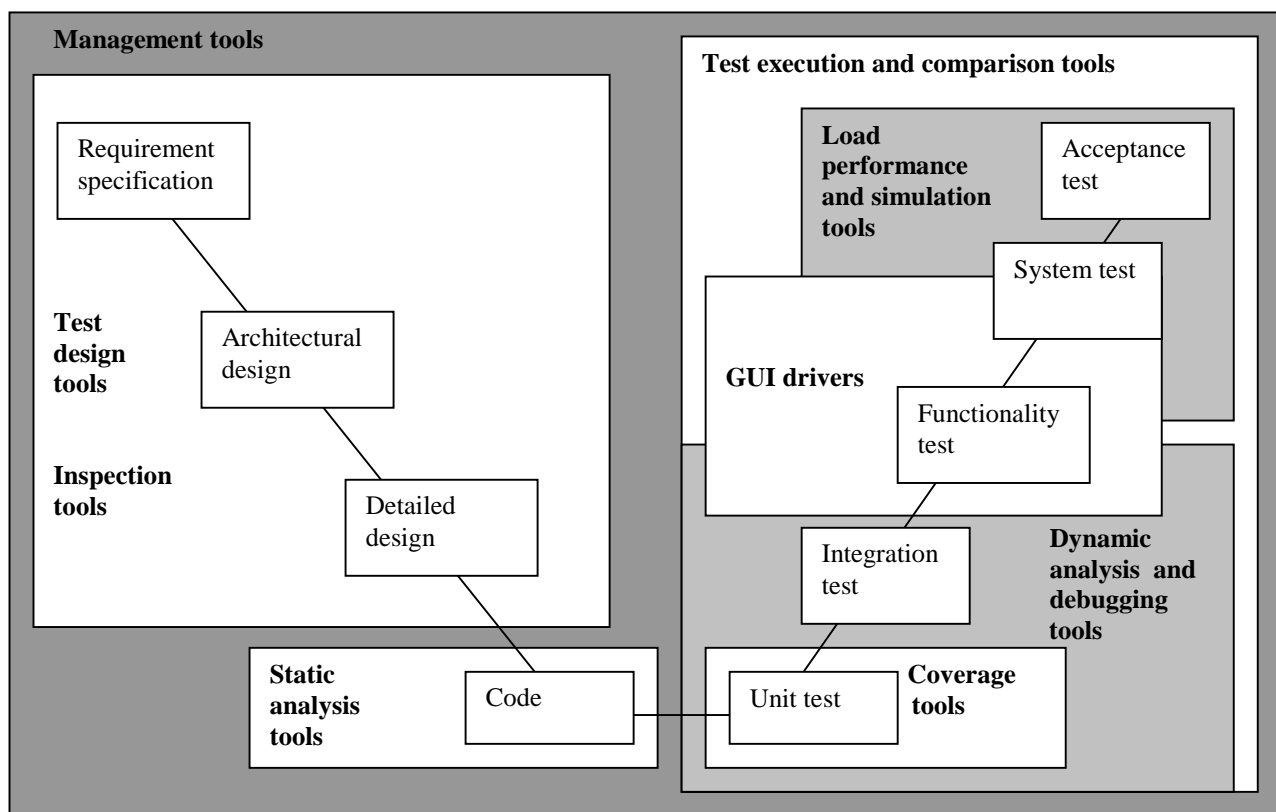


Figure 2. Division of the tools in the software development life cycle (V-model)

2 References

- [FeG99] Fewster, M., Graham, D.: Software Test Automation. ACM Press, New York, 1999.
- [Ter00] Tervonen, I.: Katselmointi ja testaus. Lecture notes in University of Oulu, 2000.

3 Division of the Tools

Every tool has a similar structure of the description. It contains firstly in the header line: Name of the tool, Company name and www-address(es). Then the description begins with one sentence, which explains the main scope of the tool. A short software description is after that and finally in the last lines possible free use and platforms of the tool if known.

3.1 Test Design Tools

3.1.1 Test Case Tools

1. Caliber-RBT, Technology Builders, www.TBI.com

Test case design system for requirements-based testing. CaliberRBT is a requirements-based, functional test case design system that drives clarification of application requirements and designs the minimum number of test cases for maximum functional coverage. By thoroughly evaluating application requirements for errors and logical inconsistencies, CaliberRBT enables project teams to refine and validate the requirements earlier in the development cycle. The earlier in the cycle requirement errors are found and corrected, the less costly and time-consuming they are to fix. CaliberRBT uses the requirements as a basis to design the minimum number of test cases needed for full functional coverage.

CaliberRBT then allows project teams to review both the requirements and the test cases in a variety of formats, including a logic diagram and structured English functional specification, to ensure that the requirements are correct, complete, fully understood and testable.

Free demo.

Platforms: Any

2. Inferno, Gresham Enterprise Software, www.gresham-software.com

Test cases creating tool. Automating with Inferno allows you to create repeatable test cases and leverage your investment throughout the lifetime of the application. Automated testing with Inferno condenses testing cycles while increasing test coverage. Inferno's capabilities include simplified script development and maintenance, creation of an intuitive library of reusable components that support shared-scripts and data-driven-scripts, and the ability to conduct unattended testing 24x7, even through system under test failures.

Free demo.

3. RadSTAR, IMI Systems Inc., [FeG99].

Test design and execution. RadSTAR is a model-based, automated software testing approach initially developed in the USA by IMI Systems Inc. It is a combination of a test planning methodology and an automated engine, which executes the planned test cases.
Platforms: Any

4. SoftTest, Bender and Associates, www.methods-tools.com/tools/testing.html

Test case design tool. SoftTest is a functional test case design tool that can be used to generate test definition scripts for any type of application, written in any language, and run on any platform. It uses a mathematically rigorous technique to verify a system's functional requirements by generating a minimum set of test case scripts designed for maximum functional coverage.

Version: 5.2

Platforms: Any

5. TDGEN, Software Research, Inc., www.soft.com/Products/index.html, www.testingfaqs.org/t-design.htm#TDGEN

Test case generation tool. TDGEN is a test data generator which works as a stand-alone product or as part of the fully integrated TestWorks/Advisor tool suite. TDGEN takes an existing test script and substitutes new random or sequential data values to create additional tests. TDGEN increases the size of a test suite to more fully exercise the application under test. TDGEN behaves as an intelligent macro processor. The user either creates new test data files or configures existing test scripts to substitute different data items for selected fields. With TDGEN, hundreds of additional tests can be created in a short amount of time.

Platforms: SPARC SunOS; SPARC Solaris ; HP-9000; DEC-Alpha OSF1; NCR 3000; DOS

6. Validator/Req, Aonix, www.aonix.com, <http://www.methods-tools.com/tools/testing.html>

Test case generator. Validator/Req performs test case generation. Producing up to 1000 test cases per minute, Validator/Req automatically extracts test specifications from project models. The Validator/Req product includes StP/T. StP 2.6 combined with Validator/Req is a suite of development tools to close the loop between requirement specification, modeling and testing.

Platforms: Sun Solaris/SunOS, HP-UX, AIX, Silicon Graphics IRIX, Digital Unix

3.1.2 Database Tools

7. DataFactory, Quest Software Inc., www.quest.com

Populate test databases with meaningful test data. DataFactory will read a database schema and display database objects such as tables and columns, and users can then point, click, and specifically define how to populate the table.

Generates meaningful data from an extensive test database that includes tens of thousands of names, zip codes, area codes, cities, states, and more.

Enables users to test with millions of rows of data, giving a realistic picture of database performance in the production environment.

Project-oriented architecture allows multiple tables and databases to be loaded in a single session.

Supports direct access to Oracle, DB2, Sybase and any ODBC (Open Database Connectivity) compliant database.

Enables developers to output to the database or to a flat text file.

Maintains referential integrity, including support for multiple foreign keys.

Gives developers the option to use DataFactory data tables or import their own data from a database or delimited text file.

3.1.3 Data Generators

8. Datagen2000, Superfine Software, www.superfinesoftware.com

Test data generator designed exclusively for Oracle databases. A level of flexibility, scalability and ease of use. The ability to harness the power of PL/SQL (Procedural Language/Structured Query Language) to define your test sets. Comprehensive support for foreign key relationships. An interactive generation environment.

Free download.

9. Datatect, Banner Software Inc, www.Datatect.com

Test data generation tool. Generate to flat files directly to database tables. Create flat ASCII files with fixed or variable length fields and records, which can be used with any appropriate application or imported into a database or spreadsheet. Directly populate tables for many popular databases, including Oracle, Microsoft SQL (Structured Query Language) Server, Informix, Sybase, and Access, using ODBC (Open Database Connectivity). The user has the capability to read in existing database table structures to aid in the generation of meaningful test data beds.

Free trial.

Platforms: Windows 95, 98, NT

10. JustData Enterprise, JustSoft Pty Ltd, www.justsoft.com.au, www.testingfaqs.org/t-design.htm#JustData

Multi Purpose Data Tool for IT people. JustData is a rapid data generation tool for IT person(s) who need to create large amounts of structured data, prior to testing applications. Working with ADO (Active Data Objects)/SQL ANSI92 compliant databases systems, General applications, Spreadsheet applications, MSSQL Server V6.0 - 7.0, Oracle 7.0 - i8.0 SQL*Loader.

Platforms: Windows 95, 98, NT, 2000

11.Move for Legacy, Princeton Softech, Inc.,

<http://princetonsoftech.com/products/comparefordb2.asp>

Move for Legacy™ is specifically designed to extract and move legacy data to support a variety of data processing tasks:

Integrate DB2 and legacy data to create accurate, consistent and realistic test data.

Take advantage of COBOL or PL/I copybook information for transforming legacy records.

Define relationships to extract all required legacy data for easy migration to a target legacy system or a relational database.

Move data from multiple databases and legacy data files without complex SQL or custom programs.

Mask sensitive data or transform legacy data for specific test scenarios.

Move for Legacy uses sophisticated algorithms to maximize its performance. Extracts that had taken hours using other approaches are done in minutes. Whether you need to create unique test databases, split or combine databases, or move data into production, Move for Legacy gets the job done with guaranteed accuracy.

3.1.4 General Test Design

12.ObjectGeode, Verolog Inc., www.methods-tools.com/tools/testing.html

ObjectGEODE is a solution for the analysis, design, verification and validation (through design model simulation), code generation, and testing of real-time and distributed applications. ObjectGEODE supports the consistent integration of complementary object-oriented approaches based on the OMT (Object Modelling Technique), SDL (object-oriented design language) and MSC (trace language) standard notations.

Version: 1.1.

Platforms: Sun Solaris/SunOS, AIX, HP-UX

13.ObjectPlanner, ObjectSoftware Inc., www.obsoft.com,

<http://www.testingfaqs.org/t-misc.htm#objectplanner>

C++ Test schedule generator tool. ObjectPlanner is a software tool to automate the generation of testing schedules. Applications written in the C++ language require unit and integration testing to reveal errors. The amount of time, number of people and other

resources required to test an application are usually not known. ObjectPlanner analyses C++ classes and calculates the approximate time that will be required to perform unit testing. This allows software developers and managers to calculate the approximate time schedules to perform unit and integration testing.
Platforms: SPARC - SunOs 4.1.X and Solaris 2.X

14. Test Designer, Intusoft, <http://www.intusoft.com/products.htm>

Test synthesis, fault analysis, isolation strategies and strategy reports. Test Designer automates test synthesis, fault analysis, isolation strategies, and generates detailed test strategy reports. It is valid for all types of system, IC (Integrated Circuit), and board-level designs. It can include analog, mixed-signal, and mechanical elements as well as behavioral and C code AHDL (Hardware Description Language) blocks. The user can easily pair different circuit configurations with various analyses to create test setups, assign measurements and define pass/fail limits for the detection of failures.
Free demo.
Platforms: Win 95/98/ME & NT 4.0/2000/XP

15. T-VEC Test Generation System, T-VEC Technologies, www.t-vec.com, www.testingfaqs.org/t-design.htm#tvec_anchor

Specification-based Test Vector Generator, Coverage Analysis and Test Driver Generation. The toolset, called the T-VEC Factory, provides specification-based test vector generation, coverage analysis and test driver generation. The Factory tools are integrated in an environment to support the development and management of structured or object-oriented requirements specifications. This process supports both unit testing, integration and software system testing through hierarchical specifications. A Specification Compiler verifies the consistency of the specification and then transforms it into a form appropriate for test vector generation.
The Test Vector Generator automatically generates tests by determining the input space boundaries for each requirement specification element, and then selects inputs along the boundaries; it uses the inputs to compute the expected output based on the specification. The Test Coverage Analyzer verifies that each requirement specification element has associated test vectors; if not, uncovered requirements are flagged. An execution/debug environment is included to aid in locating specification errors. The test execution can be fully automated from the specification-based tests.
The environment supports generation and management of specification-to-implementation objects mappings that can be customized for any implementation language and test environment. The Test Driver Generator automatically creates test drives from the object mappings to support test execution and results analysis.
The environment functionality is packaged in a visual environment that has a look and feel similar to the Microsoft WindowsNT Explorer and includes Visualization - a matrix representation of the test vectors and of artifact status.
Platforms: PC, WindowsNT

3.2 GUI Test Drivers

1. **Android, Smith House, <http://www.smith-house.org/open.html>, www.testingfaqs.org/t-gui.htm#Android**

Automated GUI Client Testing Tool. Android is an open-source GUI testing tool that can observe and record your interactions with a graphical user interface, creating a script that can repeat the session to any desired level of accuracy. The user can specify test points

where the designated window is to be checked to see if it differs from the "canonical" test run originally recorded, signalling a test failure if this is the case. It can also be used to write GUI automation scripts.

Freeware.

Platforms: Record mode requires Expect. Playback or scripting will work on any Posix-compliant system with a working port of Tcl/Tk (programming languages).

**2. AutoTester for OS/2, AutoTester Inc., www.autotester.com,
<http://www.testingfaqs.org/t-gui.htm#AUTOS2>**

Creation, execution and documentation of tests. AutoTester for OS/2 is an automated testing tool designed specifically for the creation, execution and documentation of automated tests for OS/2-based applications.

Platforms: OS/2-based apps.

**3. AutoTester for Windows, AutoTester Inc., www.autotester.com,
<http://www.testingfaqs.org/t-gui.htm#AUTWin>**

Creation, execution and documentation of tests. AutoTester for Windows is an automated testing solution designed for the creation, execution and documentation of automated tests for Windows 3.x, NT, and 95 applications. AutoTester®, a comprehensive automated testing application is specifically designed to support project teams in automating functional and regression testing.

Platforms: Windows 3.X, Windows 95, Windows NT

**4. CAPBAK, Software Research, Inc., www.soft.com/Products/index.html,
<http://www.testingfaqs.org/t-gui.htm#CAPBAK>**

Test Capture/Playback Tool. CAPBAK is a test capture/playback system, which works as a stand-alone product or as part of the fully integrated TestWorks/Regression multi-platform suite of testing tools. CAPBAK captures all user activities during the testing process using three modes: TrueTime, Character Recognition, and Object Oriented modes. Keystrokes, mouse movements, captured bitmap images, widget/object activity and extracted ASCII characters are captured into a test script. The captured images and characters provide baselines of expected application behavior against which future tests runs are compared. CAPBAK's automatic output synchronization capabilities ensures reliable playback, allowing tests to be run unsupervised as often as required.

Platforms: SPARC SunOS; SPARC Solaris; HP-9000; DEC-Alpha; NCR 3000; DOS; Win 3.x, 95, NT

**5. Certify, WorkSoft, Inc., <http://www.worksoft.com/>,
<http://www.testingfaqs.org/t-gui.htm#Certify>**

Business-user-oriented automated testing system. Certify provides enterprise-level test automation. It allows tests to span applications, platforms and test tools while shielding

users from the complexities of script languages. Certify detects application changes and automatically maps them to affected tests, simplifying maintenance and protecting the investment in the test repository.

Free demo.

Platforms: Windows 98, NT, 2000. May require third party test tool for client/server applications.

6. CitraTest, Tevron, LLC, <http://www.tevron.com>, <http://www.testingfaqs.org/t-gui.htm#CitraTest>

Automated GUI client testing tool. CitraTest is the client side Automated Testing Solution for all Citrix hosted applications.

This tool is ideal for latency, functional, and regression testing.

Platforms: Windows NT, Windows 2000, Windows 98, Windows 95, Windows ME

7. JavaStar, Sun Microsystems, www.methods-tools.com/tools/testing.html

JavaStar is an automated software testing tool for testing Java applications and applets through their GUIs. As a GUI testing tool, JavaStar compliments JavaSpec (Test Implementation Tools)- the Java API testing tool. JavaStar is created specifically for, and focused exclusively on, testing Java.

Version: JDK.

No support since 1999.

Platforms: 1.1.1. Java and JDK 1.1 platforms

8. Panorama-2, International Software Automation, Inc. (ISA), <http://www.softwareautomation.com>, http://www.testingfaqs.org/t-gui.htm#ISA_C

Capture/playback, Coverage, Test Case Minimization, Memory Check, Defect Tracing.

Panorama-2 is an enhanced product of Panorama C/C++. It is a comprehensive environment for software design, coding, testing, QA, reengineering, debugging, defect tracing, and problem management, consists of eight integrated tools (OO-Playback, OO-MemoryChecker, OO-DefectTracer, OO-Test, OO-SQA, OO-Browser, OO-Diagrammer, and OO-Analyzer), offers Object (Window) oriented and GUI-based capture/playback (without scripting and extra programming) with test coverage analysis and test case minimization so that only the minimized test cases will be played back.

Test coverage analysis of template/class/function/block/branch/ segment/loop boundary/condition & multi-condition outcome, code execution frequency analysis in function level and branch level, requirement/test case and code correspondence analysis and dynamic tracing.

Memory leak/violation checking, defect/problem tracing, quality measurement using static and dynamic OO-metrics. Automated error simulation, error checking/detecting/locating, dynamic test result mapping (from object code to source code or user-defined/system header files).

Dynamic test result display in colourful class inheritance chart, function call graph, on-line accessible reports, and logic/control flow diagrams with unexecuted path/segment/condition outcome highlighted.

Platforms: SUN Sparc, OS/Solaris, Windows NT, Windows 95, HP-UX (new).

**9. QARun, Compuware Corporation, www.compuware.com,
<http://www.testingfaqs.org/t-gui.htm#QARUN>**

GUI capture, script development test automation tool. *QARun*TM is an enterprise-wide test script development and execution tool that is part of Compuware's *QACenter*TM family of application testing products. *QARun*'s automated capabilities improve the productivity of testers, enabling them to efficiently create test scripts, verify tests, and analyze test results. Using *QARun*, tests can be easily maintained for repeat use.

With Compuware's *QACenter*, software testers and application developers can now ensure application quality with the enterprise-wide testing solution that includes client/server automated testing tools, mainframe testing tools, test process management tools and testing services. *QACenter* consists of client/server automated testing tools as well as Compuware's mainframe testing products, *QAHiperstation*TM, for VTAM (Virtual Telecommunications Access Method) applications, and *QAPlayback*TM, for CICS (Customer Information Control System) -based applications.

Platforms: All Windows and character based platforms

**10. QC/Replay, CenterLine Development Systems, Inc., www.centerline.com,
<http://www.testingfaqs.org/t-gui.htm#qcreplay>**

GUI record/replay/scripting language for X applications. QC/Replay offers "widget-based" testing of GUI based applications. Testers can record sessions and play them back on a variety of platforms. "Widget-based" rather than bitmap based verification makes test scripts robust: they are unaffected by changes in window size, position, window manager, screen size and colour, as well as changes in the application during normal development.

QC/Replay uses Tcl (programming language) for scripting, and supports portable testing across platforms.

Platforms: SunOS, Solaris, HP-UX, SVR4, UnixWare, Digital UNIX and AIX.

**11. QES/EZ for GUI, QES, Inc., www.qestest.com,
<http://www.testingfaqs.org/t-gui.htm#QESEZGUI>**

Automated GUI software testing system without scripting. QES/EZ is an automated GUI software testing system without the need for scripting or programming. The test data is stored in a relational database, which makes the maintenance and management of the test data very efficient and easy. QES/EZ for GUI has 2 run levels, extensive timing and comparison controls, reporting capabilities, variables, dynamic interaction with the target systems behaviour, etc. QES/EZ gives you the power to automatically validate, verify, test and populate any software systems without scripting. You do not have to tell QES/EZ what to test! You can instantly capture and execute the test!

Platforms: Windows 3.x, Windows/95, Windows/NT, OS/2

**12. Rational Robot, Rational Software Corp, www.rational.com,
<http://www.softwareqatest.com/qatweb1.html>**

Automated functional testing tool. Allows user to create, modify, and run automated functional, regression, and smoke tests for e-applications built using a wide variety of independent development environments. Rational TestManager and Rational SiteCheck are included, enhancing ability to manage all test assets and have them available to all team members; includes capabilities for Web site link management, site monitoring, and more. Platforms: Windows

**13. Rational TeamTest, Rational Software Corp, www.rational.com,
<http://www.testingfaqs.org/t-gui.htm#TeamTest>**

Automated test tool for Windows client/server applications. Rational TeamTest is a complete set of functional testing tools, delivering seamlessly integrated functional testing of enterprise-level client/server applications. Built on a scalable, integrated server-based test repository, Rational TeamTest combines leading-edge testing power and comprehensive management tools to set the standard for automated testing of client/server applications. Platforms: Windows NT 4.0 SP 5 or 6a; Windows 2000, Windows 98 second edition; or Windows 95 SR2

**14. SET (Solution Evaluation Tool), IBM,
<http://www.as.ibm.com/asus/svs.html>, <http://www.testingfaqs.org/t-gui.htm#SETTOOL>**

GUI test drivers and capture/replay tool, load and performance tool. The Solutions Evaluation Tool (SET) is IBM's unique, non-intrusive, automated, computer-aided software testing tool. It drives mouse, keyboard, and graphical interfaces to simulate peak work periods in complex client/server environments. SET works with any application under all operating systems in a multi vendor environment.

The SET tool can be beneficial in: Testing the performance of existing applications, testing the integration of new applications, performing server consolidation testing and evaluating e-business performance.

SET Highlights: Measures application performance and capacity, evaluates application design alternatives, validates functional specifications, determines network, server, and client workstation sizing parameters, isolates performance problems, measures actual end-to-end response times across any multi-tier environment, works in conjunction with and supports calibration of load generation tools and supports workstations, network stations, and portables.

Platforms: DOS based controller software will run on any PC with an ISA (Industry Standard Architecture) or MC (MicroChannel) architecture.

**15.Smalltalk Test Mentor, SilverMark, Inc., www.silvermark.com,
<http://www.testingfaqs.org/t-gui.htm#stm>**

Test framework for Smalltalk. Automated testing tool for VisualAge for Smalltalk and Cincom's VisualWorks. Test Mentor is a automated testing framework for Smalltalk. It seamlessly integrates UI record/playback with domain object testing for deep regression testing of your applications. Test Mentor automatically gathers execution and method coverage metrics, and provides analysis tools so you can rapidly measure the health of your project.

Platforms: All platforms supported by VisualAge for Smalltalk and Cincom's VisualWorks

**16.SQA TeamTest: ERP-Extension for SAP, Rational Software Corporation,
www.rational.com, <http://www.testingfaqs.org/t-gui.htm#TeamTestSAP>**

Functional Testing and Implementation Tool. SQA TeamTest ERP (Enterprise Resource Planning) Extension provides automated validation (functional testing) of SAP R/3 implementations. It is packaged as an SQA Robot extension, so you get the advantages of SQA Robot's object-level recording, which provides robust test scripts, even when the SAP application changes as a result of configuration changes.

Platforms: Microsoft Windows NT 3.51, Windows NT 4.0, or Windows 95, and SAP R/3

**17.SQA TestFoundation for PeopleSoft, Rational Software Corporation,
www.rational.com, <http://www.testingfaqs.org/t-gui.htm#TestFoundation>**

Functional Testing and Implementation Tool. SQA TestFoundation for PeopleSoft is an Automated Application Quality (AAQ) solution for PeopleSoft. AAQ products help ensure success in PeopleSoft implementations through predefined methodologies, steps and automatic processes.

SQA TestFoundation for PeopleSoft contains industry-proven methods for successful implementations, comprehensive requirements lists and their related validation steps, and the software solution for automated validation from Rational. Rational is the pioneer in automated testing of PeopleSoft applications and sets the standard for PeopleTools testing, with the industry's object-level integration with PeopleTools.

Platforms: Microsoft Windows NT 3.51, Windows NT 4.0, or Windows 95, and PeopleTools 6 or greater

**18.Test Mentor for VisualAge Smalltalk, SilverMark Inc.,
<http://www.javatesting.com/Product/VisualAge/VAsmtlkDev.html>,
www.methods-tools.com/tools/testing.html**

GUI record/playback and domain object (model) testing. Test Mentor for VisualAge Smalltalk is an automated testing solution designed specifically for VisualAge Smalltalk and Generator. It integrates GUI record/playback and domain object (model) testing, while providing a test architecture that strongly encourages test case reuse. SilverMark's Test Mentor exploits the VisualAge Smalltalk development environment for test case version control, and provides interfaces for test management, results analysis, and method coverage

metrics.

Version: 1.1b

Free evaluation copy.

Platforms: Win 95/98/NT, Win3.1, AIX, OS/2

19. Test Now, ST Labs, Inc., www.stlabs.com, <http://www.testingfaqs.org/t-gui.htm#testnow>

Add-on package to Microsoft Test. It has been discontinued, but is still available as a free download.

The idea of the product is made up of four parts:

1. It supplies the user with common routines that must always be created when starting a new project with MS Test. Basically, it's 10,000 lines of code the user doesn't have to write but would have had to otherwise.
2. It supplies the user with a common approach to creating their test cases. All too often people are given MS Test and told to "automate the testing of the product." Keeping things flexible to make future changes and enhancements easier is difficult to do your first couple of times automating a product. This guides the user to help keep things flexible for future needs by allowing them to incorporate a "framework."
3. A full and heavily commented suite of tests is included. Basically, it's a "real world" test suite that they can tear apart and learn from when creating their own automated tests when using Microsoft Test.
4. A stand-alone utility for not only capturing X-Y coordinates when dealing with "fake" or virtual Windows controls, but to also keep it on an independent system so that moving from one resolution to another will keep coordinate changes to a bare minimum.

The product comes with the source code for most of the routines. We do not suggest nor support changes to the code, however.

Free download.

Platforms: Windows

20. TestBench400, Original Software, www.origsoft.com, <http://www.testingfaqs.org/t-gui.htm#TestBench400>

Total test automation for the IBM AS400 (iSeries). TestBench400 - fully integrated Database, Screen & Report test automation designed specifically for AS/400 systems:

- Batch and Interactive program coverage
- Advanced Test Planning, Data Creation and Management
- Recording, Playback and Verification of Scripts - Native, GUI & web browser
- Full Test Results Analysis and Audit-Ready Reporting

Platforms: AS400

21. TestQuest Pro Test Automation System, TestQuest, Inc.,
www.testquest.com, <http://www.testingfaqs.org/t-gui.htm#TestQuest>

Automated software testing tool. TestQuest provides non-intrusive test automation tools and services for information appliances, general computing, handheld devices and industrial control. Our products, services and expertise enable you to easily automate the testing of your complex products resulting in decreased test cycle time and increased product quality and customer satisfaction.

Free demo.

Platforms: The software under test may be executing under any operating system. The TestQuest Pro runs on MS Windows 2000.

22. TestRunner, Qronus Interactive, www.qronus.com,
<http://www.testingfaqs.org/t-gui.htm#testrunner>

For automated system testing. A non intrusive automated software quality system for interactive embedded and real-time environments. Specially designed to handle systems incorporating any combination of GUI and I/O channels. (Serial, Digital, Lan, etc.)

Platforms: TestRunner runs from an independent NT host, connecting Non-Intrusively to the system under test

23. Vermont HighTest Plus, Vermont Creative Software, www.vtsoft.com,
<http://www.testingfaqs.org/t-gui.htm#vermont>

Record-Playback. Vermont HighTest Plus is an integrated tool set for automated testing of Windows 95, NT, and 3.1 applications. Capabilities include object-level record/playback of all Windows and VB controls, a Point and Click Suite Manager, a natural-syntax scripting language, automatic logging of test results, and an integrated debugger. The Suite Manager lets users combine scripts into test suites, structure scripts into hierarchies, and create nested loops -- all without programming. The intuitive, natural scripting language makes it easy to revise scripts, declare variables, perform conditional logic, set up nested loops, manage data files, and write results. Scripts can be played back unattended. Script maintenance is simplified by object-level recording, modular scripts, and hierarchical script organization.

Free evaluation.

Platforms: Windows 95, NT, and 3.1

24. Visual Test, Rational Software Corporation, www.rational.com,
<http://www.testingfaqs.org/t-gui.htm#VisualTest>

GUI Test Drivers and Capture/Replay Tools. Rational Visual Test is an automated testing tool that makes it easier and cost-effective for organizations to deploy mission-critical applications for the Microsoft Windows 95 and Windows NT operating systems and for the Web. Rational Visual Test helps developers rapidly create tests for applications of virtually any size and created with any development tool. Visual Test is integrated with Microsoft Developer Studio, a desktop development environment, and has extensive integration with Microsoft Visual C++.

Platforms: Win 3.1x, Win for Workgroups 3.11, Win 95 and NT

25. WinRunner, www.merc-int.com, <http://www.testingfaqs.org/t-gui.htm#winrunner>

Automated GUI client testing tool. Automated Testing Tool for MS Windows applications. WinRunner is an integrated, functional testing tool for your entire enterprise. It captures, verifies and replays user interactions automatically, so that you can identify defects and ensure that business processes, which span across multiple applications and databases, work flawlessly the first time and remain reliable.

Platforms: Windows NT and OS/2

26. imbus GUI Test Case Library, imbus GmbH, <http://www.imbus.de>, <http://www.testingfaqs.org/t-gui.htm#GUI-TestLib>

Library of predefined test cases for automated GUI testing. The imbus Test Case Library is a powerful and easy to use add-on for Mercury Interactive's WinRunner. It extends WinRunner's functionality and provides solutions for frequent test programming problems. The library functions help you to improve, simplify and speed up your test implementation process. They have been proven in numerous GUI testing projects.

Platforms: Runs with Mercury Interactive WinRunner (4.04 or higher) on Windows NT/WIN 95 systems.

27. XRunner, Mercury Interactive, www.merc-int.com, <http://www.methods-tools.com/tools/testing.html>

Functional test tool. XRunner is designed for testing the GUI portion of XWindows applications. XRunner 4.0's integrated testing tool components includes the RapidTest Script Wizard, which navigates all available UI paths to create a complex test script suite for unattended regression testing, and the Visual Testing environment, which combines object-oriented recording, point-and-click test generation and test script logic into a single environment, flexible verification and replay options, sophisticated reporting tools, and portability across multiple platforms.

Version: 5.0

Platforms: IBM AS/400, Sun Solaris/SunOS, HP-UX, AIX, Digital Unix

28. X-Designer, Imperial Software Tecnology, [Imperial Software Technology, www.methods-tools.com/tools/testing.html](http://www.methods-tools.com/tools/testing.html)

GUI - builder and code generator. X-Designer (XD) is a comprehensive GUI builder for Motif (an industry standard graphical user interface). It also generates code for Windows and Java. X-Designer provides built-in GUI testing for Motif applications with its XD/Replay feature. The XD/Capture feature enables existing, even hand-coded, Motif applications to be re-engineered in X-Designer and migrated to Java. No run-time licences or proprietary code are required.

Version: 5.0

Free evaluation.

Platforms: Sun Solaris/SunOS, Digital Unix, HP-UX, AIX, Silicon Graphics IRIX, DEC VMS, Linux, other Unix platforms

3.3 Load and Performance Testing Tools

- 1. AutoController with Virtual DirectTest, AutoTester Inc.,
www.autotester.com, <http://www.testingfaqs.org/t-load.htm#Directtest>**

SAP R/3-Load and Performance Tool. AutoController with Virtual DirectTest gives SAP R/3 users true end-to-end load and performance testing by simulating transactions of hundreds or thousands of users against the R/3 system.

Platforms: Windows NT.

**2. AutoController, AutoTester Inc., www.autotester.com,
<http://www.testingfaqs.org/t-load.htm#Controller>**

Load Tester/Manager. AutoController provides a centralized environment for distributed or stress testing of Windows and OS/2 client/server applications across a network.

AutoController is the automated solution for distributed testing of your Windows and OS/2 GUI and client/server applications.

From a central point, you can execute and monitor tests created with AutoTester for Windows and OS/2, across a network, for load, stress and performance testing purposes.

Any combination of tests may be scheduled for concurrent or synchronized playback across any combination of network machines.

During execution, test progress is monitored allowing complete control of each workstation. When the tests are complete, the results are immediately available for failure and performance analysis.

AutoController gives you the power and flexibility you need to put your GUI and client/server applications to the test.

Platforms: Windows 95, Windows NT, OS/2 2.1 and higher.

**3. Benchmark Factory, Quest Software, Inc., www.quest.com,
<http://www.testingfaqs.org/t-load.htm#Benchmark%20Factory>**

Load testing and capacity planning. Benchmark Factory® is a load testing and capacity planning tool for critical e-business environments. It can be used to put enormous stress on a system, which is typically hard to achieve in a standard testing environment. When a system breaks it is typically under extreme load, and this is generally when it is needed most. By identifying system capacity and performance bottlenecks before they occur, Benchmark Factory facilitates proactive testing, which in turn reduces downtime, development costs, and potential loss of revenue.

With Benchmark Factory you have the ability to:

Determine system throughput and capacity for web, database, email

Create web tests using a standard web browser, a built-in spider, or actual web traffic history

Examine tests results with built-in analysis features or Excel Reports

Simulate thousands of concurrent users with a minimal amount of hardware

Platforms: Windows NT/2000, Windows 9x/ME, Capable of testing internet applications, email servers, file servers and database servers.

**4. Chariot, Ganymede Software Inc., www.ganymedesoftware.com,
<http://www.testingfaqs.org/t-load.htm#Chariot>**

Multiprotocol networks load and performance tool. Chariot is an innovative test tool that determines the true end-to-end performance of complex, multiprotocol networks. It runs as a set of coordinated network programs that simulate the data traffic of real client/server applications. Use it to evaluate the performance of a single device - such as a router, adapter, or switch - or to measure the end-to-end performance of a network. Creating and running tests is simple, whether you want to simulate a single file transfer or create hundreds of connections, mixing protocols and types of application traffic.

Platforms:

Console platforms: WinNT and OS/2

Endpoint platforms: Win31, Win95, WinNT for x86, WinNT for Alpha OS/2, NetWare, AIX, Digital UNIX, HP-UX, Linux, Sun Solaris, Novell Netware, MVS

5. CYRANO ServerPack, CYRANO, Inc., www.cyrano.com, <http://www.testingfaqs.org/t-load.htm#CyranoL>

Automated Load and Stress Testing with RDBMS (Relational Database Management System) Analysis. Automated performance testing of client/server systems using simulated clients. ServerPack can simulate client loads for any type of client (MS Windows, X- Windows, VMS, batch processes, etc) which communicate via TCP/IP. ServerPack can also drive the running of MS Windows application on PCs.

Initial test scenarios and scripts are typically created using wizards. The CYRANO TestCommander automatically captures SQL transactions between the client and the server. ServerPack includes a powerful integrated GUI transaction modeler, which simplifies script development by allowing users to enhance SQL transactions for multi-user tests without having to write code. Test analysis includes the standard response time graphs as well as DBMS performance data, which is integrated into response time graphs. This allows you to really "drill-down" to find the links between poor response times and database issues (locking contention, table scans, etc).

Platforms: WindowsNT, Solaris, HP/UX for client simulation Microsoft Windows NT, Windows 95 and Windows 3.x for optional playback of Windows applications.

6. DataShark, Hardball Software, www.methods-tools.com/tools/testing.html

Oracle application testing tool. DataShark is an automated software quality tool that is used to test Oracle applications against realistic database models. Major features and functions include: 1) definition and creation test data from scratch; this data can be created based on business rules that you specify or can be randomly generated; 2) extraction of subsets or supersets of data from your existing databases, with the ability to automatically propagate or redefine referential integrity as needed; 3) black box testing, generating the minimal number of test cases with maximum coverage based on equivalence class partitioning and boundary condition analysis, and; 4) performance testing that allows testers to create and remove indexes on a database, along with the ability create and test SQL scripts to measure the impact of alternative keys and indexing schemes.

Version: 2.6.

Platforms: Win 95/98/NT

7. JavaLoad, Sun Microsystems, www.methods-tools.com/tools/testing.html

Load testing tool for Java programs. JavaLoad can load test an enterprise application from a pure client perspective or isolate each server-side interface and protocol. Using the JavaLoad Console, from any machine accessible to the JavaLoad test, JavaLoad generates reports of all test data, including the number of simulated users engaged, the average response time per user, an analysis of the session, and a comparison of different test sessions.

Version: JDK (Java Development Kit).
Platforms: 1.1.1. Java and JDK 1.1 platforms.

8. Load Runner Product Family, Mercury Interactive, www.merc-int.com, <http://www.testingfaqs.org/t-load.htm#loadrunner>, www.methods-tools.com/tools/testing.html

Multi-user system and server testing tool. Automated client/server system testing tools which provide performance testing, load testing and system tuning for multi-user applications

Platforms: LoadRunner/UX: for UNIX/X Window Applications, LoadRunner/PC: for MS Windows Applications, LoadRunner/XL: for Server Functionality and Performance, LoadRunner/RTE: for Terminal-Based Applications

9. preVue, Rational Software Corporation, www.rational.com, www.methods-tools.com/tools/testing.html

Quality and performance testing. preVue is a remote terminal emulator that provides quality and performance testing of character-based applications.

Version: 5.0

Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, OS/2, HP-UX, Digital Unix, DEC VMS

10. preVue-ASCEE, Rational Software Corporation, www.rational.com/products/prevue/, <http://www.testingfaqs.org/t-load.htm#preVue>

Quality assurance, performance measurement and regression testing. preVue performs quality assurance, performance measurement, and regression testing of character-based UNIX applications. The preVue system is the basis for the entire preVue product family. preVue's record/playback technology allows testers to automatically record the workload of 1 to 1000's of virtual user sessions, create test scripts, execute test sessions, and evaluate the summarized reports and graphs. preVue utilizes Remote Terminal Emulation to replicate the workload of actual application users on the System Under Test. preVue provides verification of and timing information for the System Under Test responses over communication mediums.

preVue allows testers to perform multi-user performance benchmarks that measure the application response times, system throughput, and system stress levels. preVue has no special hardware requirements and has been successfully ported to a wide range of UNIX platforms.

Platforms: Supported on all major UNIX platforms.

11. preVue-C/S, Rational Software Corporation, www.rational.com, www.methods-tools.com/tools/testing.html

Multi-user quality and performance testing. preVue-C/S provides accurate and scaleable multi-user quality and performance testing for a wide range of client/server environments. preVue-C/S supports successful deployment of client/server applications through emulation of hundreds or thousands of users for stress load testing and performance measurement.

Version: 5.0

Platforms: Win3.1, Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, OS/2, Win 95/98/NT, HP-UX, Digital Unix, MacOS, IBM AS/400

**12.preVue-X, Rational Software Corporation, www.rational.com,
<http://www.testingfaqs.org/t-load.htm#preVue-X>, www.methods-tools.com/tools/testing.html**

Regression and performance testing. preVue-X provides non-intrusive regression and performance testing of X Window applications. preVue-X accurately emulates multiple users while requiring no changes to the application or test environment.

Version: 5.0

Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, HP-UX, Digital Unix

**13.PureLoad, Minq Software AB, www.minq.se,
<http://www.testingfaqs.org/t-load.htm#pureload>**

Performance Testing and Analysing Tool. PureLoad is a Java-based performance testing and analyzing tool that helps application developers by automating the process of stress testing their server applications. This is achieved by simulating several hundreds of users performing tasks. Statistics are used to analyze the bottlenecks in the system, primarily the performance, scalability and quality characteristics.

Platforms: Java 2 version 1.3. PureLoad has been tested and verified on Solaris/SPARC, Windows NT/2000 and Linux.

**14.QALoad, Compuware Corporation, www.compuware.com,
<http://www.testingfaqs.org/t-load.htm#QALOAD>**

Enterprise-wide load testing tool. QALoad™ is Compuware's enterprise-wide load testing tool for client/server or midrange hosted applications. It supports Oracle, Sybase, Microsoft SQL Server and any ODBC (Open Database Connectivity) -compliant database, as well as http (web), character-based (TelNet), or Tuxedo traffic. QALoad can stress your client/server system by simulating hundreds or thousands of users simultaneously performing different operations. It creates test scripts that are used to simulate application transactions on the client/server system without involving end users or their equipment. QALoad is part of Compuware's comprehensive QACenter™ family of application testing products.

Platforms: All Windows and character based platforms

15. Rational Quantify, Rational Software Corp, www.rational.com

Performance testing tool. Rational Quantify for Windows automatically pinpoints performance bottlenecks in Visual Basic, Visual C++ and Java applications. It enables you to analyze performance data and make the changes that will have the greatest impact. And you can measure the results of each improvement as you tune your application for maximum performance.

Rational Quantify offers server-side as well as client Java support. With Quantify installed on your web server, you can find Java performance bottlenecks. You can run Quantify against Java Server Pages (JSPs) and Java Servlets running on Web servers such as BEA WebLogic Server or Apache Jakarta Tomcat.

Free evaluation.

16. Rational Suite TestStudio, Rational Software Corp, www.rational.com

Performance testing tool. Rational Suite TestStudio supports testing practitioners with a repeatable, extensible process within an environment of coaching and assistance that allows testers to be highly productive in finding and communicating defects to the team. With Rational Suite TestStudio, testers, developers, and analysts share the same data and metrics about the product, system, software requirements, plans, test implementations and test results. This ability to communicate clearly with each other makes it easier to keep the focus on iterative quality and unifies the entire group into a new, productive team.

Rational Suite TestStudio removes the impediments to successful testing and helps testers focus on what really matters: assuring the quality of the software being developed. Rational Suite TestStudio frees developers from the time-consuming task of trying to reproduce defects and locate their sources because testers can communicate a rich set of pinpointed information about the defects they find with point-and-click simplicity.

Free demo.

17. RemoteCog Product Family, Five Nine Solutions Inc, www.fiveninesolutions.com, <http://www.testingfaqs.org/t-load.htm#RemoteCog>

Multi-user system and server testing and monitoring framework. RemoteCog is cost-effective flexible and extensible. You can reuse all of your existing scripts and programs. You preserve your investment. If you want, over time, you can take advantage of RemoteCog features to replace hard to maintain scripts. One of the immediate benefits is being able to control and monitor your tasks from any RemoteCog Control Center. If any script or program fails, the system can page or alert anyone. It can attempt to restart the tasks or take other action.

Using the Scenario Wizard (part of the Control Center), you can define scenarios that execute a series of commands or scripts. You can create a scenario to shut down a DBMS on multiple machines and then initiate system backups and maintenance. As your needs become more complex, you can take advantage of our product features and RemoteCogs to implement your new requirements.

For instance, the Database RemoteCog can be used to create SQL scripts to perform database maintenance, shutdown and startup. You can even parameterize the SQL to handle multiple databases. Using the OS RemoteCog, you can launch programs and scripts across one or many machines. You can easily control and run different software packages from one place using one interface - the RemoteCog Control Center. This simplifies training, support, reduces the chance of errors and reduces total costs.
Free trial.

**18. SilkPerformer, Segue Software, Inc., www.segue.com,
<http://www.testingfaqs.org/t-load.htm#SilkPerformer>**

Load and performance testing tool. SilkPerformer V is a load-testing tool that provides true visual verification under load capabilities for Web applications. This allows you to capture content errors that occur only when your application is subjected to a realistic load, enabling you to greatly increase the accuracy and reliability of your system. SilkPerformer V's extensive request/response logging mechanism allows for extremely efficient root cause analysis even in large-scale load tests.

19. SilkRealizer, Segue Software, www.segue.com, www.methods-tools.com/tools/testing.html

Functional and performance testing tool. SilkRealizer is a scenario tool that enables non-technical, as well as technical users to develop and deploy true system level testing by combining functional and performance tests to simulate real world events to assure that applications will meet the requirements, before "going live".

Version: 1.2.1

Platforms: Win 95/98/NT

**20. Teleprocessing Network Simulator,
www.networking.ibm.com/tns/tnsprod.htm, <http://www.testingfaqs.org/t-load.htm#TPNS>**

Performance, function, & automated regression testing, and network design evaluation tool. Teleprocessing Network Simulator (TPNS) improves network computing by conducting stress, performance, regression, function and capacity planning tests with SNA (Systems Network Architecture), CPI-C (Common Programming Interface for Communications) and enhanced TCP (Transmission Control Protocol) support. A market leader with a track record in enterprise testing for over 20 years with an ISO 9000 certification, TPNS can be used to determine system performance and response time, to evaluate network design, to perform functional testing, and to automate regression testing. Used as a basic tool in a comprehensive test plan, TPNS increases the effectiveness of system testing by providing a structured and systematic approach to all phases of testing. Also, to accommodate your business needs, a variety of TPNS-based testing services offerings are available.

Platforms: IBM S/390, IBM S/370 processors and the following operating systems: MVS/370, MVS/XA, MVS/ESA, OS/390, and VM/ESA

21. VisionSoft/PERFORM, VisionSoft Inc., VisionSoft, Inc. , www.methods-tools.com/tools/testing.html

Performing and optimizing tool. PERFORM analyzes your application's execution behavior to identify the most executed sections for performance improvement. Statement frequency (ie. execution counts) and function/method execution time data is collected. PERFORM automatically produces reports and color highlighted source code displays to locate places where code can be optimized for performance improvement. PERFORM can apply 10 different source code optimization techniques along with cache optimization methods to yield performance improvements. PERFORM works with any C/C++ build and application execution environment.

Version: 6.3

Platforms: Sun Solaris/SunOS, HP-UX, OS/2, MacOS, AIX, DEC VMS, VxWorks, Win 95/98/NT, Win3.1, Silicon Graphics IRIX, DOS

3.4 Test Management Tools

3.4.1 CORBA

1. SilkPilot, Segue Software, Inc., www.segue.com , http://www.testingfaqs.org/t-driver.htm#SilkPilot

Functional and regression testing of middle-tier servers. SilkPilot lets you quickly and easily test the behavior of distributed objects within your application's server components.

SilkPilot can be used to test CORBA (Common Object Request Broker Architecture) servers implemented in any programming language, as well as pure Java servers through RMI (Remote Method Invocation) public interfaces. SilkPilot also explicitly supports the Enterprise JavaBeans (EJB) component model. Using SilkPilot, there's no need to build custom test programs - a simple point-and-click user interface lets you create tests without coding.

SilkPilot is offered in Standard and Professional Editions:

The Standard Edition is an entry level, which includes features for the interactive testing of objects.

The Professional Edition includes all Standard Edition features, plus powerful test automation and code generation capabilities.
Platforms: Siemens, Stratus, Win 95/98/NT

**2. Test Manager, Julian Jones Ltd, www.julianjones.com,
http://www.testingfaqs.org/t-driver.htm#Test_Manager**

Testsuite construction, execution, and management tool. TestManager is a software development system written in 100% pure Java. It provides an IDE (Interactive Development Environment) for working with regression test suites, including the facility to create, categorise, execute, and archive a set of tests.

Testcase procedures can be constructed from a set of base procedures supplied with the system. These procedures are configured through the IDE with the use of properties. No programming is required in order to execute arbitrary programs, SQL queries, or HTTP transactions within the IDE. In addition, the system provides a facility for registering custom procedures written in Java, allowing the system to be extended to execute any kind of test procedure, including testing of CORBA servers, EJBs, Java RMI or legacy systems. The system will automatically verify the execution of a test suite without the need for user intervention, providing summary statistics on the outcome of the test executions
Platforms: Any

3.4.2 C/C++

**3. Cantata, Quality Checked Software Ltd., www.qcsltd.com,
<http://www.testingfaqs.org/t-driver.htm#cantata>**

Test Harness, Coverage Analyzer, Static Analyzer tool. Cantata provides a high productivity solution for unit and integration testing of C and C++ code. It provides comprehensive facilities for DynamicTesting, Test Coverage and Static Analysis in a single integrated package.

Free demo.

Platforms: Most development and target systems including DOS, OS/2, Windows, Unix and VMS and supports most commonly used C and C++ compilers.

**4. CHILL/C/C Pilot, Kvatro Telecom AS,
<http://www.telelogic.com/index.cfm>, <http://www.testingfaqs.org/t-driver.htm#kvatropilot>**

Programmable debugger and conformance/regression tester for CHILL/C/C++ programs.

Programmable source-level thread debugger and test execution engine for programs written in CHILL, C or C++. The Pilot is designed for networked, real-time software. The Pilot can handle multiple programs, each with multiple threads. Programs need not be stopped in order to be attached. Single threads may be stopped by command or at breakpoint, without stopping the program. Interrupt handling and other processing may continue while some threads are stopped or single stepped. Pilot commands may be interpreted at breakpoints, for much higher throughput use functions called inside the program at breakpoint (trace with user-specified data, breakpoint filter, program manipulation). The Pilot is fully programmable. Scripts are written in a scripting language with expressions, 'for' and 'if' statements just like the source language (adapts to the source language of the application, if you program in C, you get C expressions and control statements.) A general macro facility (textual macros with parameters) lets you program the Pilot for complex tasks. Fast start-up and access, typically less than 5 seconds even for very large telecom applications. Minimal interference until program control services are actually used, full speed real-time execution. Platforms:

Hosts: Sparc/Solaris, SCO Unix on 486 PCs.

Targets: All hosts, and other platforms (please enquire).

5. CTA++ - C++ Test Aider, www.testwell.fi, <http://www.testingfaqs.org/t-driver.htm#cta>

C++ test harnessing tool, unit/integration testing. CTA++ (C++ Test Aider) is a tool for unit testing C++ classes, libraries and subsystems. CTA++ facilitates effective testing characterized as: easy-to-use and powerful arrangement to model the test suite into test cases, various forms of assertions for automating the test result checking, clear PASS/FAIL reporting on test cases and the whole test session, making the test runs visible, compact HTML browsable reporting of test results, regression testing, reading actual and expected values from command line or from compact textual data files, support for stub functions, reusing test cases of base class when testing inherited classes, testing multi-threaded code, testing all the advanced features of C++ (inheritance, overloading, exceptions, private parts, etc.), and more. Read more from <http://www.testwell.fi/ctadesc.html>

Platforms: Windows-2000/NT/9x, Solaris, HP/UX, Linux

6. CTB, Testwell, www.testwell.fi, <http://www.testingfaqs.org/t-driver.htm#ctb>

C test harness generator, unit/integration testing environment. CTB (C Test Bed System) generates test driver (and stubs, if needed), which are compiled with the units under test resulting in a test bed program. The test bed building can be incremental and arranged on "as needed" basis using makefiles. Using C-like command language the test bed facilitates specification-based (black-box) unit and integration testing. Both interactive and script-based tests are supported. The work becomes automated, effective, intuitive, visible, documented, standardized, measurable. Read more from <http://www.testwell.fi/ctbdesc.html>

Platforms: Windows 2000/NT/9x, HP/UX, Solaris, Linux.

7. McCabe Reengineer, www.mccabe.com

Reengineering legacy systems. McCabe Reengineer is an interactive visual environment for understanding, simplifying, and reengineering large legacy software systems. Based on twenty years experience of measuring and reengineering software applications, McCabe Reengineer provides comprehensive system analysis to locate high risk and error prone code that will form the basis of reengineering efforts. By automating the documentation of critical software characteristics you can immediately attain: faster understanding of architecture, location of high-risk code, focused development efforts and accurate resource planning. McCabe Reengineer brings focus, speed, and reliability to your reengineering process, resulting in cheaper accelerated redevelopment, with faster time to market. Supported languages: Ada, C, C++, COBOL, FORTRAN, Java, Visual Basic.

8. OSPC, Knowledge Software Ltd., www.methods-tools.com/tools/testing.html

Portability checker. OSPC (Open systems portability checker) checks applications, written in C (also supports embedded SQL), for conformance to company coding standards, International standards, and conformance to the C standard. OSPC can also flag portability problems between two platforms, such as 32 vs 64 bit differences, or using different API's. Version: 4.2c.

Platforms: Sun Solaris/SunOS, HP-UX, AIX, Digital Unix.

9. Temporal Rover, Time Rover, www.time-rover.com/Trindex.html

Scripting language tool for sequence detection and specification based verification. If you need code that detects certain sequences, such as User clicked on a banner between viewing video-A and video-B, you can write your sequence definition, using temporal logic, and use the Temporal Rover to generate the appropriate code. The Temporal Rover is a tool for the specification and verification/validation of protocols and reactive systems. The Temporal Rover will save you verification time and money, and will reduce human errors during verification. It is a tool that can automate the verification of real-time and relative temporal properties of your design (hardware or software).

The TemporalRover is a Specification based Verification tool. It lets you write formal specifications inside your source files (e.g. C, C++, Ada, Java, Verilog, etc.), within specially marked comments. The formal specification is written using a combination of Temporal Logic and your language of choice. Furthermore, the Temporal Rover converts the formal specification into executable code, to be executed when you test your program. When you do not want this code, simply do not feed your program files through the TemporalRover filter. The Temporal Rover is command line based, so you can invoke it from your Make and batch files.

10. VectorCAST, Vector Software's, www.vectors.com, <http://www.testingfaqs.org/t-driver.htm#VECTOR>

Unit Test Tool for Host and Embedded Applications. The VectorCAST products scan your Ada, Ada95, C and JOVIAL source code and automatically generate the test code necessary

to construct executable test harnesses for both host and embedded environments. Utilities are also included to construct and execute test cases, generate the reports necessary to provide an audit trail of expected and actual results, perform automated regression testing and code coverage.

Free demo.

Platforms: Solaris, SunOS, HP UX, AIX, Alpha Unix, NT/95, VMS

3.4.3 Others

11.Aegis, <http://www.canb.auug.org.au/~millerp/aegis/>, <http://www.testingfaqs.org/t-driver.htm#aegis>

Software configuration management tool for a team of developers. Aegis is a transaction-based software configuration management system. It provides a framework within which a team of developers may work on many changes to a program independently, and Aegis coordinates integrating these changes back into the master source of the program, with as little disruption as possible. Aegis has the ability to require mandatory testing of all change sets before they are committed to the repository. Tests are retained in the repository, and may be replayed later by developers, to make sure future change sets don't break existing functionality. Correlations between source files and test files allow Aegis to suggest relevant tests to developers. Bug fixes are not only required to have their tests pass on the fixed code, but they are required to fail on the unfixed code immediately before commit, to demonstrate that the bug has been reproduced accurately.

Platforms: Everything. Aegis is open source software.

12.ANVL, MIDNIGHT NETWORKS INC., www.midnight.com/, <http://www.testingfaqs.org/t-driver.htm#anvl>

Test driver for network products; network test suite. ANVL: The Automated Network Validation Library is a system for automated testing of network products. It enables network product companies to make their testing processes faster and more thorough. With ANVL, a user can rapidly and repeatably test network-based devices without the need of an analyzer or lots of equipment. ANVL facilitates negative testing by its ability to generate incorrectly formatted packets as well as correctly-formatted ones.

Platforms: SunOS 4.1 on Sun3 and Sun4, Solaris 2.3 on Sun4.

13.AutoAdviser, AutoTester Inc., www.autotester.com, <http://www.testingfaqs.org/t-driver.htm#Adviser>

Test manager and analyser. AutoAdviser manages the quality assurance process of all your mission critical software projects throughout their entire lifecycle. From requirements through production, AutoAdviser provides a central repository for organizing and managing your business requirements, tests and associated files, and test results.

AutoAdviser is more than a test manager - it is a powerful analysis facility which allows you to evaluate the readiness of your application for release into the marketplace. With

AutoAdviser, managers, business analysts, application users, testers, and developers can ensure software quality throughout the entire application lifecycle.

Central Repository:

AutoAdviser is a true workgroup solution for the use, management, and maintenance of your application test libraries. Serving as a central repository, AutoAdviser consolidates your test library components and provides test team members with access to those components. Business requirements, test plans, tests, and test results are all stored and managed from within AutoAdviser.

Test Planning:

AutoAdviser helps you plan your testing to ensure that all critical business procedures are tested and business requirements are addressed. Business requirements are stored in the repository and linked directly to your AutoTester tests. AutoAdviser displays your business requirements in a hierarchical format allowing you to quickly analyze the business process flow of your applications.

Full documentation features provide easy reference to requirement details and associated tests. Test execution and reporting can be controlled at the business requirement level for measuring test coverage. With AutoAdviser, you can ensure that each and every function of your application is adequately tested before release.

Test Organization:

AutoAdviser greatly reduces test creation and maintenance time by organizing your testing projects into hierarchical groups. From these groups, tests from current projects can be copied or grouped with tests from other projects.

For example, if you had a common navigational sequence, you would create a test once, then copy it into each test group that required navigational testing. If the navigational sequence changes, you would need to update only one test component - instead of hundreds of tests.

As your applications progress through their lifecycle, AutoAdviser provides a structured approach to the entire testing process. By combining individual tests into groups and projects, you can verify specific sections of an application, from a single dialog to an entire functional area without having to recreate entirely new test projects.

Test Scheduling & Execution:

AutoAdviser allows you to control your entire testing effort from one central location. Tests can be executed directly within AutoAdviser and can be scheduled for immediate playback or to run at a specific time in the future. For future execution, you can set a countdown timer or specify an exact date/time specification.

Change Control:

Project managers control the entire testing effort with AutoAdviser's change control features. Various levels of access rights, from report viewing to test modification to full project management privileges allow you to manage access to your test library components. AutoAdviser monitors changes made to your library and protects your testing assets by preventing users from overwriting files or modifying the same test files at the same time. AutoAdviser also produces audit trail reports that track changes in the AutoAdviser database, such as who modified a test file and when, making it easy to evaluate the status of your test library.

Quality Analysis and Drill-Down Reporting:

AutoAdviser's reporting options allow you to make informed decisions concerning the release of your applications. Instead of just producing simple pass/fail statistics, AutoAdviser offers a multitude of customizable reports that make it easy to analyze the progress of your testing and development effort.

AutoAdviser's status reports provide a snapshot of a project's current state by calculating

coverage and success of tests and requirements. To provide an early warning before project milestones are missed, AutoAdviser's progress reports measure the change in coverage and success between project test runs or dates. In addition, graphical drill-down reports give you an overall project status and allow you to quickly get more information by "drilling down" to the desired level of detail.

**14.AutoTester Client/Server for use with SAP R/3, AutoTester Inc.,
www.autotester.com, <http://www.testingfaqs.org/t-driver.htm#AUTCS>**

Test management and implementation tool. Designed for SAP R/3 implementation teams, this product provides automated unit, functional, integration and regression testing, as well as a powerful scripting language for general task automation.

AutoTester Client/Server is a comprehensive automated testing solution specifically designed to support R/3 project teams in the configuration and automated testing of SAP R/3. The industry's most advanced solution, AutoTester seamlessly handles the specific issues involved in R/3 functional testing and can be used to automate R/3 configuration control, master data input, training data input, response time monitoring, migration between R/3 clients, and other labour-intensive tasks.

Platforms: Windows 3.X, Windows 95, Windows NT

**15.CYRANO TEST, CYRANO, www.cyrano.com,
http://www.testingfaqs.org/t-driver.htm#CYRANO_TEST**

Host-based suite of testing tools for terminal-based, character cell applications.

CYRANO provides suite of testing tools available that was designed specifically for testing terminal-based applications. Because the tools run on the application host instead of a PC, the test engine can automatically synchronize test scripts to the I/Os of the application, use multiple threads (terminals), communicate & synchronize events between threads, monitor the resources the process being tested is using, and more.

The CYRANO suite of tools can be used not only for software testing, but process automation as well. Utilities include terminal monitoring/recording, script generation, test controller, statistical analyzer, screen image editor, native file comparator, and system clock simulator.

Platforms: IBM AIX, HP/UX, OpenVMS, Digital Unix, Sun Solaris, Windows NT, and more.

**16.DejaGnu, Cygnus Support, www.cygnus.com,
<http://www.testingfaqs.org/t-driver.htm#dejagnu>**

Testing framework for interactive or batch oriented apps. Tcl/Expect based testing framework designed for regression testing and embedded systems testing.

Freeware.

Platforms: Most Unix machines

17.MYNAH, Bellcore, <http://www.testingfaqs.org/t-driver.htm#mynah>

Testing environment, synchronous, asynchronous and interfaces to GUI testers.

MYNAH is an advanced software testing environment that can be used in all phases of software testing to exercise and analyze mainframe, minicomputer and workstation applications. MYNAH's high-level scripting language allows for great flexibility. MYNAH also allows simulation of an unlimited number of terminals for performance and stress testing.

Platforms: SunOS, Sun Solaris

18.OTF – On Object Testing Framework, MCG Software, Inc., www.mcsoft.com/, <http://www.testingfaqs.org/t-driver.htm#MCGSoftware>

Testing Framework for Smalltalk Objects. OTF is an easy-to-use framework for the developing; editing, keeping, sharing and running suites of tests for Smalltalk objects. Regression testing is automatic with full logging of results. Tests may be looped and conditional code executed for stress testing. While OTF focuses on testing modelling objects, there is also a simple mechanism for testing user interfaces. Extensions are easily added to OTF to expand OTF functionality and tailor OTF to site requirements. OTF is available on all the major Smalltalks.

Free trial.

Platforms: Windows, OS/2, & Unix via Visual Smalltalk; Visual Smalltalk Enterprise; Visual Works; Visual Age

19.QADirector, Compuware Corporation, www.compuware.com/qacenter, <http://www.testingfaqs.org/t-driver.htm#QADIRECTOR>, www.methods-tools.com/tools/testing.html

Test management of all phases of testing. QADirector® is a Windows-based test process management tool that is part of Compuware's comprehensive QACenter™ family of application testing products. It provides application and system managers, developers and QA workgroups with a single point of control for orchestrating all phases of testing. QADirector integrates test management with automated testing to provide a framework for managing the entire testing process-from planning and design to test execution and analysis. QADirector also enables you to make the best use of existing test assets (test plans, cases, scripts), methodologies and application testing tools.

With Compuware's QACenter, software testers and application developers can now ensure application quality with the first enterprise-wide testing solution that includes client/server automated testing tools, mainframe testing tools, test process management tools and testing services. QACenter consists of client/server automated testing tools as well as Compuware's market-leading mainframe testing products, QAHiperstation™, for VTAM applications, and QAPlayback™, for CICS-based applications.

Platforms: All Windows and character based platforms

**20.QAPlayback, Compuware Corporation, www.compuware.com,
www.methods-tools.com/tools/testing.html**

QAPlayback is a mainframe testing solution that is part of Compuware's QACenter family. QAPlayback supports the testing process of CICS applications. QAPlayback includes extension options to automatically capture and synchronize all files and DB2 activity connected with test transactions.

Platforms: Win 95/NT, Win3.1

**21.QCIT, Quality Information Systems, LLC, www.qistest.com,
<http://www.testingfaqs.org/QSi>**

Manual and automated test process management system. QCIT, Quality Control Information Tool, is an automated test management tool. QCIT consolidates, tracks and manages all functions, processes and information generated by the typical software testing organization. QCIT stores information by organization, project, and product (release version and build). It tracks the software testing process from requirement development, through test plan and test case development and execution. As testing is executed QCIT collects, stores and reports on test plan and test case pass/fail status. QCIT can also create and track defects and problem records when processes fail. If you can use a word processor and know how to navigate Windows Explorer, you are ready to start using QIS's test automation management tools.

Databases: All leading ODBC compliant databases

Platforms: Clients: Windows and NT, Server: PC, NT, UNIX

22.QES/Architect, QES, Inc., www.qestest.com, <http://www.testingfaqs.org/t-driver.htm#QESArchitect>

Complete CAST (Computer-aided Software Testing) system without scripting.

QES/Architect is a complete CAST system without the need for scripting or programming.

The testdata is stored in a relational database which makes the maintenance and management of the testdata very efficient and easy. The Architect has 4 runlevels, WYSIWYG editing, WYSIWYG prototyping, extensive timing and comparison controls, extensive reporting capabilities, variables and recoveries, dynamic interaction with the target systems behaviour, etc. QES/Architect will help organize every task necessary to manage a software production process: create specifications, schedules, personnel assignments, management information, validation criteria, data item repository, complete documentation and association map for all items in the relational database, and more. It also connects this management capacity to an engine that has the power to automatically validate, verify, test and populate the software systems that are being managed.

Platforms: PC: All platforms accessible from a PC running DOS, Windows or OS/2: like IBM 3270, AS/400, UNIX, VAX, Prime, CRAY, WANG, HP, UNISYS, TANDEM, DOS, OS/2, etc.

23.Rational Clearcase, Rational Software Corp., www.rational.com

Configuration management. Rational ClearCase is software configuration management solution that simplifies the process of change. To software teams of all sizes, it offers tools and processes you can implement today and tailor as you grow. Rational ClearCase provides a family of products that scale from small project workgroups to the distributed global enterprise, enabling you to:

Accelerate release cycles by supporting unlimited parallel development.

Unify your change process across the software development lifecycle.

Scale from small teams to the enterprise without changing tools or processes.

24. Rational TestFoundation for Windows 2000, Rational Software Corp, www.rational.com

Extension to Rational Team Test (GUI test drivers). Rational TestFoundation tests software applications for compliance with the Microsoft® Windows® 2000 Application Specification. It is ideal for independent software vendors and in-house teams developing Windows 2000 applications. Rational TestFoundation includes common tools, data, methods, metrics and documentation; and can be used as a free extension to Rational TeamTest.

Microsoft chose Rational to provide Rational TestFoundation and chose Lionbridge Technologies' VeriTest laboratories to provide services to validate that an application can be successfully used on the Windows 2000 platform.

Rational TestFoundation for Windows 2000 allows you to test compliance with the Windows 2000 operating environment as part of normal testing during development. Unlike mechanisms that require manual testing and management, Rational TestFoundation helps you build-in compliance to the Windows 2000 standard, making certification easier to obtain.

Freeware.

25. Rational TestManager, Rational Software Corp, www.rational.com

Test management tool. Rational TestManager is used to manage all aspects of testing and all sources of information related to the testing effort throughout all phases of a software development project.

- Test planning
- Test design
- Test implementation
- Test execution
- Results analysis

Rational TestManager handles scheduling, execution, and collection of results for every test that gets run under its command - functional, load, manual, integration, unit, etc. - including running third-party test tools and collating those results.

Using Rational TestManager, testers create, maintain or reference the test plan that organizes the test cases, iterations, and configurations, as well as external artifacts such as inputs from documents, models, change requests and Excel spreadsheets.

26.SDTF – SNA Development Test Facility, Applied Computer Technology, www.acomtech.com, <http://www.testingfaqs.org/t-driver.htm#sdtf>

Network test driver/manager. SNA (Systems Network Architecture) network product testing system. Provides PC-based environment for development, architectural conformance verification, load/stress and performance testing. Provides over 13,000 ready-to-run validated tests, and allows development of additional tests using test case development tools and an open API (Application Programming Interface).
Platforms: DOS, Windows 98, Windows NT

27.SMARTS, Software Research, Inc., www.soft.com/Products/index.html , <http://www.testingfaqs.org/t-driver.htm#SMARTS>

Maintenance and regression testing. SMARTS is the software maintenance and regression test system which operates as a stand-alone product or as part of the fully integrated TestWorks/Regression multi-platform suite of testing tools. SMARTS automates and simplifies the testing process by organizing tests into a hierarchical tree, providing the capability to automatically execute all or a subset of tests, and generating a variety of reports based on test results. The tests can be supplemented with activation commands, comparison arguments and pass/fail evaluation methods. When executed, SMARTS performs the pre-stated actions, runs a difference check on the output against the baseline, and accumulates a detailed record of test results. Six reports are automatically generated from test outcomes: Latest, All, Regression, Time, Failed and Summary reports.
Platforms: HP-9000; HP-UX; DEC-Alpha; NCR 3000; DOS; Win 3.x/ NT/ 95

28.STW/Regression for Windows, Software Research, www.soft.com/Products/index.html, www.methods-tools.com/tools/testing.html

Regression testing tool. STW/Regression for Windows automates and manages tests on both text and GUI applications. It includes CAPBAK/MSW, SMARTS/MSW, and CBDIFF, which are also available separately. CAPBAK/MSW records and plays back test sessions, capturing all user activities (keystrokes and mouse movements) during the testing process. SMARTS/MSW organizes tests into a hierarchical tree, allowing automatic execution of tests. SMARTS/MSW performs pre-stated actions and runs a difference check on outputs against the baseline. CBDIFF compares bitmap images, while discarding extraneous discrepancies during the differencing process.
Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, HP-UX, DOS

29.STW/Regression, Software Research, www.soft.com/Products/index.html, www.methods-tools.com/tools/testing.html

Regression testing tool. STW/Regression automates and manages tests on both text and GUI applications. It includes CAPBAK/X, SMARTS, and EXDIFF, which are also available separately. CAPBAK/X captures all user activities during the testing process and offers

automatic output synchronization and test case preview and optimization. SMARTS automates testing by organizing tests into a hierarchical tree, allowing automatic execution of all or a subset of tests, and reporting status, history, regression, and certification reports. EXDIFF verifies bitmaps captured during a recording session and automatically compares them with actual images at playback. EXDIFF can also determine a successful test based on actual values, using Optical Character Recognition and translating the values to ASCII characters. STW/Regression includes automated load generation for multi-user applications, and employs a test management component for automated test execution and management to allow unattended test runs.

Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, HP-UX, DOS, Digital Unix, SCO Unix, Solaris X86

30.TCM (Test Case Manager), Pierce Business Systems,
jupiter.drw.net/matpie/PBSystems/products/Development.html,
<http://www.testingfaqs.org/t-driver.htm#TCM>

Organizes test cases for storage and execution logging. Test Case Manager (TCM) is a tool designed for software test engineers to organize test cases for storage and execution logging. Test cases are written up in a standard format and saved into the system. Test cases can be organized by level (Smoke, Critical Path, Acceptance Criteria, Suggested), by area (GUI breakdown, installation, data, etc.), by status (pass, fail, untested, etc.), or other breakdown criteria. Once test cases are built, testers use TCM to track and report success or failure of test cases. TCM provides an unlimited number of central, multi-user databases, each of which will support an entire test team. TCM is intended for use by small to midsize software development companies or organizations. Most features are implemented as intuitive wizards for users to step through.

Freeware.

Platforms: Windows. Requires Microsoft Access.

31.TEO, GAKO Informatique, www.gako.fr, <http://www.testingfaqs.org/t-driver.htm#teo>

Test case management tool. TEO is a central database, that helps your validation team to share information, such as:

- Test specification (test case, test groups, test specification, test chapter)
- Test environment (product, product's version, product's supported configuration)
- Test execution (test logs, defects)
- Test planning (any tests levels, any versions and configurations)
- TEO is a test reporting tool, that helps your organization, it may produce document such as:
 - Test specification (document, statistics)
 - Test execution (report, statistics)
 - Test Campaign (preparation, follow-up, final status, statistics)

Platforms: Windows 95/98/NT/2000, Linux

**32. Test Mentor – Java Edition, SilverMark, Inc., www.javatesting.com,
<http://www.testingfaqs.org/t-driver.htm#TestMentorJava>**

Java component, unit and function test automation tool. A functional test and test modelling tool for Java developers & QA Engineers to use as they develop their Java classes, clusters, subsystems, frameworks, and other components, either deployed on the client or the server during unit and integration testing.

Free trial.

Platforms: Client (user-interface) runs on Windows platforms only, test execution on all Java platforms

**33. Test Station, AutoTester Inc., www.autotester.com,
<http://www.testingfaqs.org/t-driver.htm#Teststation>**

Capture, execution and maintenance. Test Station is an automated testing solution for the creation, execution and documentation of automated tests for character-based applications running on PCs, mid-range computers and mainframes.

Test Station provides you with the power of automated testing through:

- Flexible test capture
- Unattended test execution
- Full test reusability and maintainability
- Automated documentation and reporting
- Scripting capabilities

Platforms: Mainframe

34. TestDirector, Mercury Interactive, www.merc-int.com, www.methods-tools.com/tools/testing.html

Planning, executing and quality control. TestDirector is a scalable test management tool for planning, executing, and communicating quality control during the entire development process. It supports: 1) test case design, with definition of key steps and expected results for each test case; 2) test creation, for both manual tests and scripts for automated test execution; 3) organization and management of automated and manual test execution; 4) bug tracking, including association of bugs with tests and monitoring of bug fix times, and; 5) summary statistics of the debugging and testing activities.

Version: 5.0

Platforms: Win 95/98/NT, OS/2, Win3.1

**35. TestExpert, Silicon Valley Networks, www.svnetworks.com,
<http://www.testingfaqs.org/t-driver.htm#TestExpert>, www.methods-tools.com/tools/testing.html**

Test case organization, test execution and documentation. TestExpert is an open solution offering development teams integration with and control over their existing testing environment. TestExpert provides a powerful Oracle database repository and parallel

execution engine that organizes test cases and test information, drives high-speed test execution, and captures all results and journal information for complete IEEE (Institute of Electrical and Electronic Engineers) standard audit trail documentation and requirements traceability. TestExpert's open architecture provides "Best-of-Class" solution through seamless integration with testing tools from industry leaders, such as PureAtria, Rational, Mercury Interactive, and Segue and allows custom integration with existing "in-house" systems.

Platforms: Solaris, SunOS, HP/UX, NT

36. Testify, Applied Testing and Technology, www.about-testify.com, <http://www.testingfaqs.org/t-driver.htm#Testify>

Distribution, configuration, and execution of automated software testing. In today's marketplace, software testing - particularly Web application testing - must span a variety of platforms, operating systems, browsers, and hardware configurations. Testify is a test management tool that choreographs the distribution, configuration, and execution of automated software testing across Internet and intranet networks and collects test results into a single comprehensive report. Whether you have 5 test systems or 500, as your testing needs grow Testify grows with you.

Speed time to market: test your software across varied environments with the touch of a button.

Slash QA costs: you will quickly recoup the benefits of your investment in testify by reducing the costs of managing your tests.

Rev up testing productivity: testify managed tests can be run over the course of product development to detect and kill bugs early.

Drive consistent test procedures: use the same process each time testing is performed.

Supercharge product quality and profitability: better test procedures and productivity equals better products. Now there's more time to focus on documentation, installation, compatibility and other key issues!

Testify works with your existing tests, test processes and tool sets.

Platforms: Any Java platform: Windows 95, 98, NT, 2000, most versions of UNIX.

37. TestMaster, Teradyne Inc., [Teradyne, Inc. Software & Systems Test, www.methods-tools.com/tools/testing.html](http://www.methods-tools.com/tools/testing.html)

Test program generation system. TestMaster is an automatic test program generation system targeted to the specific needs of advanced telecommunications systems. TestMaster algorithmically generates tests in minutes using a reference model of the software's behaviour. The model is developed using graphical tools from either system requirements or a functional software specification. It can easily be updated or expanded as specifications evolve, and new tests can be generated again in minutes. The resultant TestMaster model represents a virtual data base of every possible test for the target application from which tests can be generated "on-demand" to meet specific needs ranging from basic "new build" tests to exhaustive regression tests.

TestMaster can drive virtually any automated test execution environment - either a custom tool or commercial harness. Some of the commercial test harnesses currently being programmed with TestMaster output include QA Partner, XRunner and MYNAH along with call generators such as Hammer and Crescendo.

Version: 1.8

Platforms: Sun Solaris/SunOS, HP-UX, Digital Unix

38. TestWorks, Software Research, www.methods-tools.com/tools/testing.html, <http://www.testingfaqs.org/t-driver.htm#Teststation>

Testing tools suite. TestWorks is an integrated suite of testing tools accessible from a GUI or command line. TestWorks comprises four product bundles (see separate listings) that work independently or together: STW/Regression automates test execution for GUI and client/server applications; STW/Coverage measures how much of your code has been tested in branch, call-pair, and path coverage; STW/Advisor uses 17 metrics to analyze an application's data, logic, and size complexity; STW/Web supports development and maintenance of web applications. TestWorks for Windows includes STW/Regression for Windows and TCAT C/C++.

Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, HP-UX, DOS, Win3.1, Win 95/98/NT

39. TETware, The Open Group, <http://tetworks.opengroup.org>, <http://www.testingfaqs.org/t-driver.htm#tetware>

Distributed Multi-Platform Test-Suite Manager. TETware is a multi-platform test scaffold for supporting both non-distributed and distributed test suites. It is provided as a supported source code product and supports both UNIX operating systems and the Windows NT operating system. A Java API is also available.

This product allows production of test suites sharing a common interface, therefore promoting sharing of test suites within organisation as well as between different organisation. Standardisation of the test methodology and tools allows testing efforts to focus away from the harness and tools, thus potentially increasing efficiency and productivity.

TETware is being used in a wide diversity of automated testing applications, ranging from standards API conformance testing, performance and stress testing, verification of secure electronic transactions, to distributed cross platform applications.

Platforms: UNIX®, Win NT, 95 and 98

40. TEWS (Test Tool for Windows), MSG Software Oy, www.msg.fi/fin/tuotteet/main.htm

Unit, Integration and System testing tool. TEWS is an efficient tool for defining the test specifications for a system under construction and for managing test information in software projects of all size.

TEWS allows the user to concentrate on productive work - no more test cases written on notepads. TEWS systematizes testing. TEWS helps in the planning of future projects.

TEWS lends a new dimension to quality monitoring, allowing software project testing histories to be generated for project groups or individual product categories. TEWS ensures real-time testing documentation.

TEWS is intended for software production and quality departments.
Quality departments - (software) project quality monitoring.
Design managers – ensuring the total quality of software production and projects.
Project managers - monitoring and analysis of testing.
Project groups and testers – enhancing test planning, more specific test specifications with concrete test cases, software product and component quality assurance.
Platforms: Windows 95/98/NT

**41. TMS, TESTMASTERS, www.testmastersinc.com,
<http://www.testingfaqs.org/t-driver.htm#TMS>**

Suite of Test Management Tools. The TMS (Test Management System) suite consists of three products: TPS (Test Planning System), TCS (Test Control System) and PRS (Problem Reporting System). TPS creates sophisticated Test Plans. TCS is designed to manage test data for complex mainframe systems with multiple interfaces. PRS provides a solution to problem management. Each product can be purchased individually or all in the complete suite.

Platforms: PC based tools, run in LAN environment. Windows 3.1, 95, NT.

42. TOOTSIE: A High-end OO Development Environment (Robert V. Binder), <http://www.rbcs.com/>, <http://www.rbcs.com/>

Tootsie (Total Object-Oriented Testing Support Environment) provided comprehensive test automation for BigFoot development. BigFoot is an environment developed to support testing of a large client/server application. BigFoot is a multi-platform client/server system implemented in C++. Clients use Microsoft Foundation Classes for Windows 95 and Windows NT. Server objects wrap a relational DBMS running on multiple Sun/Solaris servers. An ORB (Object Request Broker) provides client/server communication using a distributed object model. BigFoot processing modes include high volume batch, real-time external links, and GUI-based interactive sessions. The system must provide 7x365 availability. Any incorrect output could have very serious and costly consequences.

Platforms: Win95, NT, Sun/Solaris servers

**43. Validator SC, B-Tree Systems Inc., <http://www.testquest.com/>,
www.methods-tools.com/tools/testing.html**

Compatibility testing tool. Validator SC is an automated system compatibility testing tool for conducting qualitative tests on PC components in a non-intrusive, OS independent manner. Validator SC is used to test virtually any PC component, including motherboards, graphics accelerator boards, or chipsets for compatibility with any software application.

Version: 3.0

Platforms: Win 95/98/NT

3.5 Test Implementation Tools

3.5.1 Java

1. **AQtest, AutomatedQA Corp., <http://www.automatedqa.com/>, <http://www.testingfaqs.org/t-impl.htm#AQtest>**

Automated support for functional, unit, and regression testing. AQtest automates and manages functional tests, unit tests and regression tests, for applications written with VC++, VB, Delphi, C++Builder, Java or VS.NET. It also supports white-box testing, down to private properties or methods. External tests can be recorded or written in three scripting languages (VBScript, JScript, DelphiScript). Using AQtest as an OLE (Object Linking and Embedding) server, unit-test drivers can also run it directly from application code. AQtest automatically integrates AQtime when it is on the machine. Entirely COM-based (Component Object Model), AQtest is easily extended through plug-ins using the complete IDL (Interactive Definition Language) libraries supplied. Plug-ins currently support Win32 API (Application Programming Interface) calls, direct ADO (ActiveX Data Objects) access, direct BDE (Borland Database Engine) access, etc.

One Test Log for the Life of the Project - You use a test tool for the results. In AQtest all test results, and even test errors, go to one log. The log begins when you define a test project

and ends when you close it, perhaps many months later. Tens of different tests, hundreds of iterations, thousands of results all are posted to the log as messages. There are five types of messages, five priorities for each. Messages can be filtered by date, time, test, type, priority, then automatically formatted into clear, to-the-point test reports that tell you exactly what you want to know about how your application is doing. The reports are in XML for reading on any machine equipped with IE 5.0 or later.

Capture and Storage of Object Properties, Screen Clips and Output Files for Comparison - The basis of testing is evaluation. The way to automate this is to get results and compare them to verified results already achieved, or created for the purpose. AQttest can automatically capture results as collections of selected properties from an object and its children (such as a dialog and its controls), as pictures from the screen or from some other source, or as files of any type, from any source. The supplied library then makes most comparisons between "new" and "standard" the business of one or two lines of code.

Language-Independent Scripting - The design philosophy behind AQttest is to never inhibit or restrict a development team or a QAteam. One illustration of this is that with AQttest you never have to use a proprietary scripting language to code test cases. AQttest is the only tool of its kind with built-in support for multiple scripting languages: VBScript, DelphiScript, or JScript, plus C++ in recording mode. (Recorded C++ scripts can be run from inside applications, using AQttest as an OLE server.)

Scripting by Recording or by Coding - Build your test scripts quickly, and build them with the power to thoroughly test your business process. Use any mixture of recorded actions, hand-written code, calls to AQttest's powerful test library, and script tuning in the full-featured editor and debugger -- whatever fits the need, whatever is easiest for the user. Everyone in your organization can start using AQttest immediately for his or her job.

Intelligent Script Recording - AQttest's recorder goes much beyond what the world's best macro recorder could do. It records selections, string input, checkings/uncheckings, etc., on Windows controls, identified by parent, class, caption, etc. The recorded script is compact, it "speaks" the way the developer sees things, and it remains valid after interface tweaks. But that is only the beginning. The intelligence goes much deeper than Windows screen objects. AQttest recognizes typical UI library objects for your development tool. In fact, it can recognize "from outside" even the most application-specific objects if the application is compiled for this. On the other hand, the recorder can also log single keyboard events and single mouse events at absolute screen positions, with total detail, if that's what you need.

Testing that Reaches Beyond the Black Box - Can't access functionality you know is in your application? Ever wanted to execute application methods from scripts? AQttest lets you go into the application under test like no other tool can. Open Applications are applications linked with one AQttest file (supplied in source code, of course). This gives scripts access to most of their public elements -- objects, properties, methods. And, if the Open Application is compiled with external debugger information AQttest's Debug Info Agent™ can use it to watch and manage even the application's private elements, just like the IDE (Integrated Development Environment) debugger can.

Debug Info Agent™ is an exclusive AQttest technology, and an industry first. It gives "external" testing more access to application internals than even the application source code has, from other modules. "External" testing can watch or run anything in the application, just as easily as it watches and runs the application's user interface -- including calling methods and changing property values. Easy answers for the tough questions.

Object Browser - AQttest's Object Browser is divided into two panes like the Windows Explorer. In the left pane it displays a tree of all objects (processes, windows, etc.) that exist in a system, with full expand-collapse capacities. In the right pane, it displays all the available properties and methods of the object selected in the left pane. This right-pane view can itself expand its details in order to dig into the object's properties. You can also select objects directly onscreen with the mouse, and get the same detail analysis. The Browser serves two purposes. First, it lets you orient yourself among all the objects that your test scripts will have access to. Second, it lets you capture and store any collection of properties, or the image of any onscreen object, for later comparison with test output. Before storage, the specific properties to keep can be selected, or the image to save can be trimmed as needed.

Automated Self-testing - The entire process of "digging into" an application from the outside can be reversed. Again by linking in just one file, any application can gain complete control of the AQttest engine (an OLE server). Anything a script can do, application code can do, in the application's own programming language. In other words, the application can test itself, embed recorded scripts, use AQttest's scripting tools to feed itself input and to capture its own output, use the AQttest library and the result sets stored by AQttest to analyze its own behavior, and post its conclusions to the project log just like any other test would. It can also call and execute external AQttest scripts if that is useful. Self-testing applications are an obvious way to simplify Unit testing. Moreover, self-testing code is written in application source language, has the same access to application internals as the rest of the source code, and can be debugged using the same IDE debugger.

Entirely COM-based, Entirely Open Architecture - AQttest is a major automated testing tool that is entirely COM-based. It is the first because this is not easy to do. But the advantages are many. For instance, because the AQttest engine is an OLE server, it can be run from any COM-based language interpreter, or from application code. It can also have an application log its own details to it, so that it becomes "open" to external tests. Because the engine "sees" everything through one interface, COM, it is source-language independent, and can even read debugger information and use it at runtime through the Debug Info Agent(tm). Another advantage is that the various internal interfaces of AQttest are defined in IDL libraries that are supplied with the product, and which you can use to extend it. In fact, the coming Java and .NET support are such plug-in extensions.

Fully customizable interface - The onscreen interface of AQttest is clear, attractive and intuitive, by design. And it is flexible. Everything (everything) can be customized in a few seconds to adapt it to your particular needs of the moment.

Platforms: Windows 95, 98, NT, or 2000.

2. AssertMate for Java, Reliable Software Technologies, www.rstcorp.com, <http://www.cigital.com/news/java98.html>, www.methods-tools.com/tools/testing.html

AssertMate is a system that aids Java engineers in safely and accurately placing software assertions within their Java programs. Software assertions assist in finding bugs earlier in the development process (when they are easier and cheaper to fix). As developers move toward object-oriented design and component reuse, concepts such as "design by contract" and "property-based testing" use assertions to ensure proper implementation of classes,

component interfaces, and internal variables. Until now, assertions have been missing from the Java development environment.

AssertMate provides fast and accurate assertion capability for programmers and class level testers alike. AssertMate enables developers to make use of pre-conditions, post-conditions, and data assertions to validate behavioral correctness of Java programs, while providing a simple system for placing assertions without modifying source code.

Free download.

Platforms: Win 95/98/NT

3. JavaSpec, Sun Microsystems, www.methods-tools.com/tools/testing.html

JavaSpec is an automated software testing tool for testing Java applications and applets through their APIs. JavaSpec compliments JavaStar - the Java GUI testing tool. JavaSpec enables very rigorous and thorough unit testing of Java code.

Version: JDK.

No support since 1999.

Platforms: 1.1.1. Java and JDK 1.1 platforms.

4. JSUnit, Maintained by Edward Hieatt, <http://www.edwardh.com/jsunit/>, <http://www.testingfaqs.org/t-impl.htm#JSUnit>

Unit testing framework. JsUnit is a unit testing framework for client-side JavaScript. It is essentially a port of JUnit to JavaScript.

Freeware.

Platforms: Windows NT, Windows 2000 and Windows 95 (IE only). JavaScript 1.4 or higher (Internet Explorer 5.0 or later and Netscape 6.0 or later).

5. Junit, JUnit.org, <http://www.junit.org/>, <http://www.testingfaqs.org/t-impl.htm#JUnit>

A regression testing framework used by developers who implement unit tests in Java.

Open source software.

Freeware.

3.5.2 C/C++

6. APROBE, OC Systems, www.ocsystems.com

Development and testing tool. Aprobe is a software development and testing tool that lets developers and testers write probes that alter the behaviour of a program, monitor its behaviour, or log data. What are probes? Probes, or patches, provide a way to alter the image of the executable while it is in memory.

Unlike virtually every other tool on the market, Aprobe works by modifying the executable while it is in RAM. No permanent changes are made to either the source or the executable.

C/C++ supported.

Platforms: Windows 2000/NT 4; Solaris 2.5.1 / SunOS 5.5.1; Sparc & UltraSparc. AIX 4.2

7. BoundsChecker, Compuware Numega, <http://www.numega.com/>, <http://www.testingfaqs.org/t-impl.htm#boundchecker>, www.methods-tools.com/tools/testing.html

Bounds and memory leak detector. BoundsChecker detects invalid Windows API parameters, invalid Windows API return codes, invalid ANSI C parameters, invalid ANSI C return codes, memory leaks, resource leaks, dynamic memory overruns, stack memory overruns, data and heap corruptions, memory locking problems, null pointer manipulation, and processor faults.

Free evaluation.

Platforms: DOS, Windows (include. Windows 95)

8. Inuse, ParaSoft Corporation, www.parasoft.com, <http://www.testingfaqs.org/t-impl.htm#Inuse>

A graphical tool of memory problems. Inuse helps prevent several common memory problems by displaying and animating in real time the memory allocations performed by an application. An Insure++ (Evaluation tools) add-on. Programs linked with Insure++ can connect at runtime to Inuse, a dynamic memory visualization tool. Inuse is available as an add-on module to Insure++. Inuse displays valuable statistics regarding the amount of dynamic memory in use, memory fragmentation, sizes of allocated blocks, and the number of calls to memory management routines.

Platforms: Windows NT/2000; UNIX; DEC; HP; IBM; Linux, SGI and Solaris

9. ITEX , Telelogic, http://www.iec.org/exhibits/telelogic_02/, www.methods-tools.com/tools/testing.html

ITEX is a toolset for test suite development and execution based on TTCN (Tree and Tabular Combined Notation). The toolset supports test suite auditing, generation (with Telelogic SDT (Software Design Tools)), analysis, simulation (with Telelogic SDT), and test application generation in C. ITEX (Functions used in the test phase of the project) is part of the Telelogic Tau environment.

Version: 3.2

Platforms: Win 95/98/NT, Sun Solaris/SunOS, HP-UX

10.MTE, Integrisoft, www.integrisoft.com

Unit testing tool for C applications. Based on HindSight's combination of static analysis and dynamic testing, MTE relieves the developer of the burden of writing a driver and stub(s) by automatically generating them. In fact, MTE goes further by automating the entire module testing process.

Platforms: Windows NT/95/98/2000.

**11.ObjectTester, ObjectSoftware Inc., www.obsoft.com,
<http://www.testingfaqs.org/t-impl.htm#objecttest>**

C++ Unit Testing Tool. ObjectTester is a software tool to automate the generation of C++ unit test scripts. Software written in the C++ language is made up of data structures. In C++, a "class" represents a data structure and is made up of data and methods. ObjectTester analyzes C++ classes and generates unit test cases for them. The test cases generated are in the C++ language and are free from syntax or semantic errors. A unit test script is generated for each class. This script can be modified by developers to include any "pre" or "post" conditions for each method.

Platforms: SPARC - SunOs 4.1.X and Solaris 2.X.

12.Rational VisualTest, Rational Software Corp, www.rational.com

Functional testing tool. Rational Visual Test® 6.5 is an automated functional testing tool that helps developers and testers rapidly create tests for Windows applications created with any development tool. Rational Visual Test is integrated with Microsoft Developer Studio, a desktop development environment, and has extensive integration with Microsoft Visual C++.

Rational Visual Test makes developers and testers more productive and makes it easier for organizations to deploy applications of virtually any size for the Microsoft Windows 95, Windows 98, Windows NT and Windows 2000 operating systems and for the World Wide Web.

Platforms: W95, 98, NT, 2000

3.5.3 Others

**13.Access for DB2, Princeton Softech, Inc.,
<http://www.princetonsoftech.com/>, <http://www.testingfaqs.org/t-impl.htm#access>**

Modify, insert or delete data in test database. To fully test your applications, it is necessary for your test database to contain specific test cases. Some of those test cases may include data errors or combinations of data values that are rarely found in your production database. With Princeton Softechs Access for DB2, the relational editor, you can inspect your test database, modifying, inserting, or deleting data as necessary.

14.Autotest, Racal Electronics Group, [FeG99].

Automatic test harness tool for integration testing. Autotest controls a series of test cases on different programs and software units in Visula.

Each test is defined by its input data, instructions on how to run the test, and the expected output data. The tool requires that the programs to be tested are capable of being driven solely from data held in files.

Test process produces output data and then comparison process compares this and expected output data producing test report.

Autotest can automatically re-run all failed tests at the end of a set of tests, to overcome problems of unexpected events such as network failures.

15. Compare for DB2, Princeton Softech, Inc.,
<http://princetonsoftech.com/products/comparefordb2.asp>

Unit testing, integration testing and regression testing tool. It captures a snapshot image of your data “before” and “after” you run application and then automatically identifies the differences. Automating the comparison process allows you to analyze the results and resolve problems. Princeton Softech's Compare for DB2™ analyzes complex sets of referentially intact data and quickly identifies any differences — no matter where they occur in the chain of tables.

16. Complite File Comparison Family, James Veale,
<http://world.std.com/~jdveale>, <http://www.testingfaqs.org/t-impl.htm#complite>

Word-by-Word ASCII File Comparison Utilities. Word-by-Word comparison utilities for programming language source, ASCII text and data files. Native versions of these utilities are available for OS/2 Warp, Windows NT, and DOS. All versions support split screen displays with correlated scrolling and word highlighting, hardcopy and HTML printouts. Keyword and phrase processing can tailor the comparison for specific applications. Platforms: OS/2 Warp, Windows NT, DOS

17. DateWise FileCompare, DateWise, Ltd., www.datewise.com/mt,
<http://www.testingfaqs.org/t-impl.htm#datewise>

Automated token-by-token text and binary file comparison utility. Patented file comparison utility designed for both binary and text files with expected differences without requiring detailed file specification. Current versions support combinations of ASCII, EBCDIC, big endian, and little endian files. Automatically compares report files with varied dates/formats in the header (ignoring expected differences in the dates, while flagging unexpected differences, even in the heading dates), compares executables ignoring timestamps embedded by the compiling/linking process (telling the user what values were found for the timestamps within the files), etc.

DateWise's unique technique calculates the difference between two files, telling the user if the files matched and what the difference was between the files or where any unresolvable differences are located, just by providing the names of the two files. Furthermore, this tool does not require explicit delimiters (such as spaces or punctuation) to appear around the words or tokens contained in the text or binary files (unlike competitive word-by-word comparison utilities). The powerful technique is not a "silver bullet" because it uses a publicly known technology for producing its results (Patent No. 6,236,993 covers the technology the tool is based on. Other patents are pending.).

Platforms: MS-DOS, Windows, HP-UX, Solaris, and OS/390, but the tool was written in ANSI-C for portability to other platforms and we are willing to port it elsewhere.

**18.EXDIFF, Software Research, Inc., www.soft.com/Products/index.html,
<http://www.testingfaqs.org/t-impl.htm#EXDIFF>**

Extended Differencing Tool. EXDIFF is the extended file differencing system which operates as a stand-alone product or as part of the fully integrated TestWorks/Regression multi-platform suite of testing tools. EXDIFF extends commonly available file-comparison utilities by comparing files of various logical structures. This includes not only ASCII and binary files, but also bitmap image files saved either with TestWorks/Regression's capture/playback system, CAPBAK, or the standard X Window Dump utility (xwd). Used in conjunction with its TestWorks/Regression companion tools CAPBAK and SMARTS, the testing process is completely automated.

Platforms: SPARC SunOS; SPARC Solaris; HP9000; DEC-Alpha; NCR 3000; DOS; Win 3.x/ NT/ 95

**19.FREstimate, SoftRel, <http://www.softrel.com/>,
<http://www.testingfaqs.org/t-impl.htm#FREstimate>**

Software reliability prediction tool. A software reliability prediction is an "assessment of quality and reliability of the software engineering system performed prior to that system being constructed".

This software reliability prediction tool is used early in development, as early as the concept phase to predict the delivered or fielded failure rate or MTTF (Mean Time to Failure) of a software system. The software reliability prediction methods are based on historical data from similar previously fielded software projects in which the actual MTTF, failure rate or reliability is known.

Free demo.

Platforms: Windows 95, at least 100MZ processor, at least 24MB RAM and at least 15 MB available disk space.

**20.HeapAgent, MicroQuill Software Publishing, www.microquill.com,
<http://www.testingfaqs.org/t-impl.htm#HeapAgent>**

Memory error detection, including leaks, overwrites, invalid references. HeapAgent is an interactive memory error debugging tool. Detects a broad range of heap and stack errors and reports more error information than any other tool. Includes tools for browsing live heap information to diagnose the cause of errors. No relinking/recompiling, works with debuggers, and causes minimal runtime slowdowns.

Free trial.

Platforms: Windows 3.x, Windows NT, Windows 95

**21.InCtrl5, PC Magazine, <http://www.zdnet.com/pcmag/utilities/>,
<http://www.testingfaqs.org/t-impl.htm#InCtrl5>**

Monitor changes made by install programs. InCtrl5 is the fifth incarnation of one of PC Magazine's most popular utilities. By monitoring the changes made to your system when you install new software, it enables you to troubleshoot any unexpected problems that come up. Virtually every modern program uses an install utility that installs or updates files, and also may record data in the Registry, and update .ini files or other essential text files. A companion uninstall utility should precisely reverse the effects of the install utility. When a newly installed program causes existing applications to fail, or when the supplied uninstall utility can't complete its task, to restore your system you need a record of exactly what the original install utility did. InCtrl5 can provide this record.

Freeware.

Platforms: Windows 9x, NT, 2000, or Me

22.McCabe TestCompress, www.mccabe.com

GUI-based environment to affect a comprehensive test of your software systems. McCabe TestCompress is an interactive GUI-based environment for identifying the minimal set of data necessary to affect a comprehensive test of your software systems. Based on McCabe's advanced parsing technology, McCabe TestCompress has been proven to reduce test files by over 97%. By curtailing the size of complex test data files you can immediately: conserve testing resources, efficiently modify test-data, focus data extraction and test systems rapidly and more efficiently.

**23.MDBDiff, Pierce Business Systems,
<http://jupiter.drw.net/matpie/PBSystems/products/retired/MDBDiff.html>,
<http://www.testingfaqs.org/t-impl.htm#MDBDiff>**

Access database structural comparison tool. MDBDiff is a tool designed to quickly locate structural differences between two Microsoft Access databases (*.mdb files). The comparison is made on the following objects: tables, fields, indexes, relations, and queries. The structural differences analyzed include: changes, additions, and deletions of objects and properties of those objects. The user follows the intuitive wizard-like interface to select two Access databases to compare, enters any necessary security logon information, select the objects to compare, then selects an output type and location for the report that is built. Users can save settings to a text file for auto execution via command line. NOTE: the author has marked this tool as "retired".

Freeware.

Platforms: Windows (running Microsoft Access 97 or Access 2000)

24. Move for DB2, Princeton Softech, Inc., <http://www.princetonsoftech.com/>, <http://www.testingfaqs.org/t-impl.htm#move>

Automatically gathers related rows from multiple tables. Princeton Softech's Move for DB2, the Relational Copy Facility, automatically gathers related rows from multiple tables and copies these complete sets of relationally intact data. With Move for DB2, virtually all of the effort of creating test databases is eliminated, the quality of the test data base process is improved significantly, and programmers spend more time programming and testing.

25. mpatrol, Graeme Roy, <http://www.cbmamiga.demon.co.uk/mpatrol/>, <http://www.testingfaqs.org/t-impl.htm#mpatrol>

Malloc debugging library. The mpatrol library can be used to diagnose heap errors and locate memory leaks. It is freely distributable under the GNU (An operating system) General Public License and comes with full source code and comprehensive documentation.

Freeware.

Platforms: Various UNIX, Windows, AmigaOS

26. The Heap Analyst for DOS, StratosWare Corporation, <http://www.stratosware.com/swc/>, <http://www.testingfaqs.org/t-impl.htm#HA10>

Library tool. An informational utility library that shows runtime memory use statistics. The Heap Snapshot feature logs the state of the heap to a disk file, including the size, status, and address of each block, and summary statistic information for the heap. Easy one-call access to statistics like total memory free, total memory used, largest free block size, number of free/used blocks, and more. The Heap Analyst provides a heap fragmentation index which measures how efficiently the runtime library's allocation routines are managing memory. Adds less than 3K in overhead to the application size.

Platforms: CPU: Intel X86 / OS: MS-DOS / Compilers: Microsoft, Borland, Watcom

27. ZeroFault, The Kernel Group, Incorporated, www.tkg.com, <http://www.testingfaqs.org/t-impl.htm#zf>

Memory analysis tool. ZeroFault is the next generation in active memory analysis tools. Using patent pending technology, ZeroFault detects and reports run-time memory problems before they become expensive. ZeroFault works on any AIX executable without recompilation, or relinking.

Free evaluation.

Platforms: RISC/6000 AIX versions 3.2.x, 4.1.x, 4.2

3.6 Test Evaluation Tools

3.6.1 Java

1. **TotalMetric for Java, Reliable Software Technologies, <http://www.cigital.com/news/java98.html>, www.methods-tools.com/tools/testing.html**

Software metrics tool. TotalMetric for Java is a software metrics tool to calculate and display cyclomatic complexity, level-of-effort, and object-oriented metrics for the Java language. Using TotalMetric for Java during development provides an effective means of identifying risk areas and where to focus testing as you create your test plan. TotalMetric captures structural and psychological complexity metrics for Java programs and helps to assess "where risks are" in code. TotalMetric is invaluable for intelligently planning and focusing dynamic testing efforts on critical areas of code. Metrics results can be sorted by classes and methods in a fully customizable HTML analyzer. Version: 1.0.

Free download.

Platforms: Win 95/98/NT.

2. **DeepCover, Reliable Software Technologies, www.methods-tools.com/tools/testing.html**

DeepCover provides test coverage analysis for C/C++ and/or Java applications. There are two products: DeepCover for C/C++ and DeepCover for Java. DeepCover measures method, branch, condition/decision and multiple condition coverages with minimal intrusion into the development or testing process. Results are reported in four levels of granularity:

Application; Method; Class; and Condition. DeepCover supports most UNIX and Win32 compilers and the Microsoft Developer Studio IDE.

Version: 2.0.4

Platforms: Win 95/98/NT, Sun Solaris/SunOS

3. **JavaScope, Sun Microsystems, www.methods-tools.com/tools/testing.html**

Test coverage tool. JavaScope is a software testing tool that quantitatively measures how thoroughly Java applications and applets have been tested. JavaScope is created specifically for, and focused exclusively on, testing Java. JavaScope integrates seamlessly with both JavaStar and JavaSpec.

No support since 1999.

Version: JDK.

Platforms: 1.1.1. Java and JDK 1.1 platforms.

4. Rational PureCoverage, Rational Software Corp, www.rational.com

Code coverage analysis tool. Rational PureCoverage® for UNIX is code coverage analysis tool designed to be used by developers and testers during daily unit tests. Whether PureCoverage is used with Rational Purify or as a standalone tool, it is unsurpassed for ease of use and flexibility. With a single click, you can take advantage of an annotated source view that provides line-by-line code-coverage analysis, revealing the exact lines of code not covered by your test suite. PureCoverage helps you develop fast, reliable code.

Rational PureCoverage for Windows helps you to identify untested code quickly, so you can be sure you've checked your entire application for potential problems, not just part of it. An essential tool for Visual Basic®, Visual C++®, Java™, VB.NET, or C# applications, PureCoverage for Windows will speed testing efforts, save precious development time and enable you to develop fast, reliable code.

Free download.

5. TCAT for Java, Software Research, Inc., www.soft.com/Products/index.html, <http://www.testingfaqs.org/t-eval.htm#JAVA>

Test Coverage Analysis Tool. TCAT for Java, a component tool of the TestWorks/Web suite, is a test coverage analysis tool configured specifically for Java applets and for use on Java-enabled browsers. Developers of animated Web sites can use TCAT for Java to determine that their Java applets are fully exercised by their test suites -- a critical quality verification when Java applets support financial transactions on the Web. TCAT for Java provides the programmer with a concise summary of the logical conditions executed and not executed during test runs of the program. TCAT for Java does this by instrumenting every conditional statement in the program, identifies the true and false paths, and produces execution time reports -- all automatically.

Free demo.

Platforms: SPARC SunOS; SPARC Solaris; HP-9000; DEC-Alpha; NCR 3000; DOS; Win 3.x/NT/95

6. [VisualTestCoverage, \(IBM\), <http://www.alphaworks.ibm.com/tech/visualtest>](http://www.alphaworks.ibm.com/tech/visualtest)

Test coverage tool. VisualTestCoverage is a technology for determining testing adequacy for programs written in VisualAge for Smalltalk, Generator, and Java. When the programmer is testing a VisualAge application, the tool counts how many connections, events, methods of the program were executed, and displays this information in a view that resembles the VA composition editor. It provides feedback to the developer about whether he has done enough testing on each class. It covers all elements within those classes, and not just the methods.

Platforms: Windows 95

3.6.2 C/C++

7. Cantata++, Quality Checked Software, www.qcsltd.com, <http://www.testingfaqs.org/t-eval.htm#cantatapp>

Code Coverage Analyzer and Test Driver for C++. Cantata++ provides dynamic testing and test coverage analysis for C++ applications. A second generation tool in the Cantata testing tool family, Cantata++ measures statement, branch (decision), Boolean expression, call, call pair, and MC/DC (Multiple Condition/Decision Coverage) coverages as well as OO coverage measures such as inheritance and template instantiations. Cantata++ also maintains occurrence counts for number of times each measured item is encountered.

With a specific design goal of minimal intrusion into the development and testing process, Cantata++ is an effective tool for C++ unit and integration testing. Cantata++ now brings professional testing within the reach of all C++ developers and provides full support for the testing of C++ including: private members, abstract classes, templates and exceptions, all without the overhead of writing stubs. Increasing the productivity of the testing process by quickly identifying any untested statements, decisions, functions, or C++ classes, Cantata++ has been specifically designed to offer developers a high-productivity solution to the verification of C++ software in a unique, integrated toolset.

Free demo.

Platforms: Windows 95, Windows NT, Unix

8. C-Cover, Bullseye Testing Technology, www.bullseye.com, <http://www.testingfaqs.org/t-eval.htm#c-cover>, www.methods-tools.com/tools/testing.html

Test Coverage Analyser for C/C++. C-Cover increases testing productivity and saves time by quickly identifying untested control structures, functions, C++ classes, source files, and sub-systems. C-Cover is very easy to use and includes advanced features.

Platforms: NT, Windows 9x, Unix

9. CTC++, Testwell, www.testwell.fi

Test Coverage Analyzer for C/C++. CTC++ is a powerful instrumentation-based tool supporting testing and tuning of programs written in C and C++ programming languages. Everyone working with C or C++ is a potential user of CTC++.

CTC++ is available in two sale packages:

"CTC++": using the tool on a host environment. The tool utilities and the instrumented code under test are run on the selected host.

"CTC++ Host-Target & Kernelcoverage": in addition to using CTC++ normally on the host environment, cross-compiling the instrumented code for a target, running tests on the target, getting the test results back to the host, and viewing the coverage reports on the host. This package facilitates also measuring coverage from the kernel mode code on the host.

CTC++ facilitates:

Measuring test coverage => ensure thorough testing, you know when to stop testing, etc.

- Function coverage (functions called).

- Decision coverage (conditional expressions true and false in program branches, case branches in switch statements, catch exception handlers in C++, control transfers).

- Statement coverage (statements executed; concluded from function and decision coverages).

- Condition coverage (elementary conditions true and false in conditional expressions).

- Multicondition coverage (all possible ways exercised to evaluate a conditional expression).

Searching execution bottlenecks => no more guessing in algorithm tunings.

- Function execution timing (if needed, the user may introduce his own time-taking function for measuring whatever is interesting).

- Execution counters.

Conveniently presented test results.

- Hierarchical, HTML-browsable coverage reports.

- Pure textual reports.

Ease of use.

- Instrumentation phase as a "front end" to the compilation command => very simple to use.

- Can be used with existing makefiles.

Usable "in the large".

- Instrumentation overhead very reasonable.

- You can select what source files to instrument and with what instrumentation options.

- Besides full executables, libraries and DLLs can also be measured.

Integrated to Microsoft Visual C++ Developer Studio.

- See description of the CTC++/VC++ integration.

Good management and visibility of testing.

- Easy to read listings (textual and HTML).

- In terms of the original source code.

- Untested code highlighted.

- Various summary level reports.

- TER-% (test effectiveness ratio) calculated per function, source file, and overall.

10.GCT, Testing Foundations, <http://www.testingfaqs.org/t-eval.htm#gct>

C test coverage. The Generic Coverage Tool (GCT) is a freeware coverage tool that measures how thoroughly tests exercise C programs. It measures branch, multicondition, loop, relational operator, routine, call, race, and weak mutation coverage for C programs. See <http://www.testing.com>
Freeware.
Platforms: Most UNIX

11. Insure++, Parasoft Corporation, www.parasoft.com, <http://www.testingfaqs.org/t-eval.htm#Development>

Runtime error detection, memory monitor, coverage analysis tool. Insure++ is a powerful automatic runtime error detection tool for C/C++ that improves software quality, accelerates time to market, and reduces development costs. Insure++ automatically detects errors in your code, regardless of whether you know they exist. Insure++ identifies compile-time, runtime and algorithmic errors, including memory corruption, operations on uninitialized, NULL, or "wild" pointers, memory leaks, errors that allocate and free dynamic memory, operations on unrelated pointers, and more. Insure++ includes a graphical "Malloc Monitor" that helps developers understand how their programs manipulate memory and a total coverage analysis tool that provides information about which pieces of the code have been tested to insure that all of the program is bug-free, not just the most frequently used parts. Platforms: SunOS, HP, IBM RS6000, SGI 5.X, Digital UNIX, Lynx, Linux, SCO

12. TCA, Parasoft Corporation, www.parasoft.com

Test coverage tool. TCA is a tool developers can use to determine the coverage of Insure++. Using TCA with Insure++ gives developers a comprehensive bug-detection package. TCA reports which sections of code Insure++ tested, how much of the code was tested, and how often Insure++ executed each section. With TCA, developers can increase the efficiency and completeness of their testing, resulting in cleaner, more reliable code. TCA provides coverage information on three levels. The Overall Summary shows the percentage of the total application that has been tested. The Function Summary displays the coverage of each individual function in an application. The Block Summary breaks down the coverage by individual program statement blocks. One of TCA's most important features is its ability to test code in logical blocks. Most coverage analysis tools work on a line-by-line basis, a limitation that lengthens the testing process. Testing by block reduces the amount of data the developer needs to analyze and reveals which paths have been executed in addition to which statements.

13. ObjectCoverage, ObjectSoftware Inc., www.obsoft.com, <http://www.testingfaqs.org/t-eval.htm#objectcov>

C++ branch coverage analysis. ObjectCoverage is a software tool to help automate and improve the effectiveness of C++ software testing. Software written in the C++ language consists of "selection" and "iteration" statements. Selection statements are the "if", "else", "switch", and the "case" statements. Iteration statements are the "for", "while", and "do-

while" constructs. ObjectCoverage analyzes these statements and generates a test case/branch coverage report.

Platforms: SPARC - SunOs 4.1.X and Solaris 2.X

14. Panorama C/C, International Software Automation, Inc., www.softwareautomation.com

Test coverage tool. QA and maintenance toolkit for PC Windows. Segment (branch) test coverage analysis, loop boundary test coverage analysis, automatic structure analysis and diagramming, automatic source code control flow analysis and diagramming, complexity analysis and automatic document generation.

International Software Automation, Inc. (ISA) is offering a shareware version of Panorama C for Windows 3.1 product for everyone to try before purchasing a product license.

Panorama C for Windows 3.1 is a sub-set of Panorama C for Windows 95/NT. Panorama C for Windows 3.1 is NOT a freeware product. After installing the shareware version of the product in a personal computer, you can use it within 30 days without a license. After the 30 day period of time, you should choose either to purchase a product license from International Software Automation, Inc. (ISA) or an ISA's product distributor, or to delete it from the computer without using it anywhere.

Panorama C for Windows 3.1 supports MSC 6.0, MSC 7.0 and Borland C 3.1 compilers.

15. Panorama C/C++, International Software Automation, Inc. (ISA), www.softwareautomation.com, http://www.testingfaqs.org/t-eval.htm#ISA_T

Coverage, Metrics, Test Case Minimization, Error Check/simulation/Detection.

Panorama C/C++ is a comprehensive software testing environment consists of OO-Analyzer, OO-Browser, OO-Diagrammer, OO-SQA and OO-Test (the key tools for test evaluation), offers:

Test coverage analysis of template/class/function/block/branch/ segment/loop boundary/condition & multi-condition outcome,

Code execution frequency analysis in function level and branch level,

Requirement/test case and code correspondence analysis and dynamic tracing, test case efficiency analysis and test case minimization for intelligent playback (command line),

Quality measurement using static and dynamic OO-metrics,

Automated error simulation,

Error checking/detecting/locating,

Dynamic test result mapping (from object code to source code or user-defined/system header files),

Dynamic test result display in colourful class inheritance chart, function call graph, on-line accessible reports, and logic/control flow diagrams with unexecuted path/segment/condition outcome highlighted.

Free versions (for a program less than 1501 lines) of Panorama C/C++ can be directly downloaded from Panorama C/C++ web site.

Platforms: SUN Sparc OS/Solaris, Windows NT, Windows 95, HP-UX (new).

16.QC/Coverage, CenterLine Software,
www.uniforum.org/news/html/dir.open/24665.html

Code coverage and analysis tool. QC/Coverage is a code coverage and analysis tool for quickly identifying how much of an application's code was exercised during testing. By identifying code components that have not been adequately tested, QC/Coverage helps users focus their efforts, avoid wasting time, and make better use of testing resources.

QC/Coverage also provides industry-standard testing metrics to assist in test planning, with specialized support for C++. QC/Sim, a companion product, simulates situations such as a network crash or full disk that are difficult to duplicate in reality.

Platforms: DEC Ultrix, HP-UX, IBM AIX, Novell UnixWare, OS/2, Sun Solaris 2.x, SunOS

17.STW/Coverage, Software Research, www.soft.com/Products/index.html,
www.methods-tools.com/tools/testing.html

Test coverage tool. STW/Coverage measures how much of your code has been tested. This coverage analyzer does branch and call-pair coverage in a single test run. It provides full support for all standard constructs and dialects of C and C++, using logical branch, function call and path class coverage. STW/Coverage includes TCAT C/C++, TCAT-PATH, and T-SCOPE, each of which is available separately. TCAT C/C++ uses recursive descent compiler technology to produce textual and graphical reports on logical path and branch completeness at the individual function level and at the system level. TCAT/Ada works on Ada and FORTRAN programs, measuring the number of times logical branches have been exercised for both True and False conditions. S-TCAT works on Ada and FORTRAN programs, measuring the consistency of function-call pairs at the system interface level. TCAT-PATH determines how many of the possible path combinations in a module are exercised by grouping paths into equivalence classes, including multiple instances of iterations. T-SCOPE is a test data observation tool that visually demonstrates logical branches and function calls being exercised while tests are running.

Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, HP-UX, DOS

18.TCAT C/C++ for Windows, Software Research,
www.soft.com/Products/index.html, [www.methods-](http://www.methods-tools.com/tools/testing.html)
[tools.com/tools/testing.html](http://www.methods-tools.com/tools/testing.html)

Test coverage tool. TCAT C/C++ for Windows provides measurements to determine how much code has been tested. TCAT C/C++ produces textual and graphical reports that identify what has and has not been exercised. It uses logical branch coverage for unit and system testing, and function call coverage for integration tests. TCAT C/C++ uses recursive descent compiler technology to handle all standard constructs and dialects of C and C++. Free demo.

Platforms: Win3.1, Win 95/98/NT

19.T-SCOPE, Software Research, Inc., www.soft.com/Products/index.html,
<http://www.testingfaqs.org/t-eval.htm#TSCOPE>

Test Data Observation Tool. T-SCOPE, a test data observation tool, works directly with TCAT and S-TCAT, part of the fully integrated TestWorks/Coverage testing suite. T-SCOPE dynamically depicts logical branches and function-calls as they are exercised during the testing process. Slider bars can also be selected that will show the percentage of coverage achieved for individual modules or the entire program, as each test is run. Color annotation indicates how logical branch and function-call execution correspond to the minimum and maximum threshold values.

Platforms: SPARC SunOS; SPARC Solaris; HP-9000; SGI Irix 5.3, 6.2; DEC-Alpha; NCR 3000; SCO/UnixWare 2.1.1; DOS; Win 3.x/NT/95

20. VisionSoft/TEST, VisionSoft Inc., www.methods-tools.com/tools/testing.html

Code coverage and metrics tool. VisionSoft/TEST is complete code coverage tool that works with any C/C++ build and application execution environment. TEST is a source code based code coverage analysis tool. Statement, branch/limits and edge/path flow coverage analyzes are all supported. In addition VisionSoft/TEST generates application, and file metrics, McCabe complexity analysis and numerous reports including HTML based. Also as your code changes and to prevent redundant retesting, the ChangeSmart feature automatically performs version/change analysis on your code and applies the still valid coverage analysis data to the new application files.

Version: 6.3

Platforms: Win 95/NT, Win3.1, Sun Solaris/SunOS, HP-UX, OS/2, MacOS, AIX, DEC VMS, VxWorks

3.6.3 Others

21. WhenToStop, SoftRel, <http://www.softrel.com/>, <http://www.testingfaqs.org/t-impl.htm#WhenToStop>

Software reliability measurement tool. This software reliability tool is used during testing or once there are observed failures and to estimate whether or not the required or predicted failure rate or MTTF objective will be met. Hence it is a software reliability estimation tool. This tool uses observed data to estimate software reliability. The software can help you determine when to stop testing and can also help you determine the resources needed for maintaining the software. Software reliability estimation tools can also be used to validate software reliability prediction tools. The software can also help you determine the time needed to reach an objective failure rate.

22. CodeTEST, Applied Microsystems Corporation, www.amc.com, <http://www.testingfaqs.org/t-eval.htm#codetest>

Family of software verification tools for embedded systems.

CodeTEST is software verification tool suite specifically designed for embedded systems software. CodeTEST traces embedded programs and verifies software performance, test coverage, and memory allocation. CodeTEST can monitor an entire program at once, eliminating the difficulty of deciding what part of the program to watch, and how to set up

the tool for each measurement. It can produce reliable traces and measurements when programs are executed out of cache or are dynamically relocated.

The CodeTEST tool suite includes:

Performance Analysis

- Measures function and task execution times
- Counts call-pair linkages to identify thrashing
- Non-sampled measurements of 32,000 functions at one time

Coverage Analysis

- Displays coverage at program, function, or source levels
- Plots coverage over time
- Interactive measurements simplify test creation and refinement

Memory Allocation Analysis

- Dynamic display shows memory leaks in progress before system crashes
- Pinpoints memory allocation and free errors to offending source line
- Measures true worst case allocation

Trace Analysis

- Traces embedded programs at source, control-flow, or high (task) level
- Deep trace captures over 100 thousand source lines of execution
- Powerful triggering and trace display options zero in on problems

Platforms: MS Windows 95, Sun Sparc, HP 9000

**23.Hindsight/TCA, Integrisoft, www.integrisoft.com,
<http://www.testingfaqs.org/t-eval.htm#hindsight>, www.methods-tools.com/tools/testing.html**

Test Coverage Analysis. Provides five levels of test coverage. Both high and low level results can be displayed on structure charts, and source code diagrams. The results can also be formatted in 16 different reports. Hindsight/TCA also provides path analysis, identifying conditions that need to be satisfied to reach untested areas.

Platforms: SunOS, Solaris, HP/UX, and IBM AIX

**24.Hindsight/TPA, Integrisoft, www.integrisoft.com,
<http://www.testingfaqs.org/t-eval.htm#hindsight2>**

Test Case Analysis. Hindsight/TPA analyzes the contribution of each test case. Reports are generated which show:

1. Which test cases should be run on modified code.
2. How efficient each test case is.
3. Which test cases are redundant and overlapping.

Hindsight/TPA also cross references the test cases with the functions and source code.

Platforms: SunOS, Solaris, HP/UX, and IBM AIX

25.LDRA Testbed, LDRA Ltd, www.ldra.co.uk, <http://www.testingfaqs.org/t-eval.htm#LDRA%20Testbed>

Static Analysis and Code Coverage Toolset. Analyses source code statically to enforce coding standards & calculate metrics on quality, complexity, & maintainability. Static Analysis also features coding error detection, data flow analysis, variable cross reference, & code visualisation. Dynamic Analysis measures code coverage of statements, branches, test paths, subconditions, & procedure calls.

Analyses: Ada, Algol, C/C++, Cobol, Coral 66, Fortran, Pascal, PL/M, PL/1, Assemblers (Intel + Motorola).

Platforms: Solaris, SunOS, Windows 95/NT/3.1x, Vax/VMS, Open VMS, HP-UX, Digital Unix, WinOS/2, AIX, IRIX, SCO, MVS.

26. Logiscope, Verilog Inc., <http://www.telelogic.com/>, <http://www.testingfaqs.org/t-eval.htm#Logiscope>, www.methods-tools.com/tools/testing.html

Quality and Productivity for the development, test and maintenance of software applications

1) LOGISCOPE TestChecker: Measures the completeness of tests implemented on your application regardless of whether performed on host or target platforms in terms of source code test coverage. Highlights areas of non-executed code (i.e., not checked) during tests.

- Displays test coverage "on-line", i.e., during the execution of the test (even when it is performed on a target platform)

- Identifies effective non regression if program files have been modified

- Identifies inefficient test (regarding test coverage level)

- Identifies test conditions required to test unreachable parts of the application,

- Record and re-execute tests cases automatically

2) LOGISCOPE RuleChecker: Checks coding rules, naming conventions and presentation rules. It allows you to select rules from the set of 50 pre-established rules provided with the tool, or introduce rules specific to your company, reused from one project to another.

3) LOGISCOPE ImpactChecker: It proposes a request system for checking the use of resources. These requests are relevant to location, type of data used and dependencies between data.

4) LOGISCOPE Audit: is a graphic tool used to evaluate program quality. It provides users with relevant diagnosis based on computation of a set of software metrics (more than 90).

Maintenance teams can rely on graphic overviews to understand the program logic and architecture.

Platforms: UNIX: Sun, HP, IBM, Digital / WINDOWS: NT and 95

27. McCabe Visual Testing Toolset, McCabe and Associates, www.methods-tools.com/tools/testing.html

McCabe Visual Toolset quantifies the number of tests required for full path coverage and generates the associated test conditions at the unit and integration levels. The product will also monitor the testing and report on code coverage and list the untested paths.

Version: 5.11

Platforms: Win3.1, Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, DEC VMS, Win 95/NT, Digital Unix, HP-UX

28.SofInst, SES Software-Engineering Service GmbH,
<http://www.testingfaqs.org/t-eval.htm#sofinst>

Test Coverage Tool. SofInst is a test coverage tool for measuring mainframe COBOL and PL/I programs with CICS, DLI, SQL or DDL/DML macros. On the basis of the instrumentation technique, the programs can be instrumented at different 5 levels of instrumentation. The coverage report shows which probes were executed how many times together with a coverage ratio.

Platforms: IBM/MVS Mainframe

29.S-TCAT, Software Research, www.soft.com/Products/index.html,
www.methods-tools.com/tools/testing.html

Test coverage tool. S-TCAT is a system test coverage analysis tool that can work alone or with the STW/Coverage tools. S-TCAT works on Ada and FORTRAN programs, measuring the consistency of function-call pairs at the system interface level.

Version: 8.2

Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, HP-UX, DOS, SCO Unix, Solaris X86

30.TCAT-PATH, Software Research, Inc.,
www.soft.com/Products/index.html, <http://www.testingfaqs.org/t-eval.htm#TCATPATH>

Path Coverage Analysis Tool. TCAT-PATH is a path coverage analysis tool that works as a stand-alone product or as part of the fully integrated TestWorks/Coverage tool suite. TCAT-PATH analyzes a large number of module paths (sequences of logical branches) by grouping paths into equivalence classes, including multiple instances of iterations. For critical modules, path coverage can be used to extend TestWorks/Coverage's TCAT logical branch coverage at the unit test level. While TCAT's C1 metric determines how many logical branches in a module are exercised, TCAT-PATH's Ct metric determines how many of the possible path combinations in a module are exercised.

Platforms: SPARC SunOS; SPARC Solaris; HP-9000; DEC-Alpha; NCR 3000; DOS; Windows 3.x; MSWindows NT; MSWindows 95

31.Test Center, Centerline Software, www.centerline.com,
<http://www.testingfaqs.org/t-eval.htm#testcenter>

Memory access anomaly detection and coverage tool. TestCenter helps programmers thoroughly test their code early in their development process. TestCenter not only helps programmers detect run-time errors, but also helps them identify areas for further testing through a graphical code coverage analyzer. TestCenter is an ideal add-on to all programming environments.

Platforms: Sun, HP, AIX

3.7 Static Analysis Tools

3.7.1 Java

1. CMTJava - Complexity Measures Tool for Java, www.testwell.fi

CMTJava, Complexity Measures Tool for Java, is a code metrics tool for Java language. It is intended for mature software development organizations striving for productive development process resulting in high quality products.

CMTJava Facilitates:

Calculating the basic code complexity metrics:

- McCabe's cyclomatic number.

- Halstead's software science metrics.

- Lines of code metrics.

Extremely fast processing:

- CMTJava can measure many thousands of Java source lines in a second.

Usage in the large:

- In one CMTJava session you can analyze a single file, a couple of files, or your whole application possibly containing hundreds of thousands of code lines.

Independent of the used Java development environment:

- CMTJava does not use, not even assume the presence, of a Java development environment. The tool only assumes that it can read the Java code from normal operating system source files.

Further processing of the measurements:

- CMTJava can write the measures into a text file, which is readily suitable input for Excel (or any other spreadsheet program), by which the measures can be processed further, say for various company summaries or for graphical representation.

Ease of use:

- CMTJava is outmost simple to use from the command line.

- Includes thorough on-line helps, man pages in Unix.

Platforms: Windows 2000/NT/9x and many Unix environments

2. CodeCompanion, Jenssoft, www.jenssoft.com, <http://www.testingfaqs.org/t-static.htm#CodeCompanion>

A software analysis tool that enforces coding conventions on Java source code.

CodeCompanion checks Java source code against a set of carefully selected coding convention rules, and reports whether it complies to the rules or not.

Free demo.

Platforms: CodeCompanion is written entirely in Java, and is therefore available for all platforms with a Java 2 Virtual Machine.

3. JavaPureCheck, Sun Microsystems, www.methods-tools.com/tools/testing.html

JavaPureCheck is a static testing tool, which reads class files and reports everything which might cause applications or applets to be unportable. JavaPureCheck is the official purity checker testing tool of JavaSoft's 100% Pure Java Initiative.

Version: 4.0.

Freeware.

Platforms: Java and JDK 1.1 platforms.

4. JStyle, (Man Machine Systems), <http://www.mmsindia.com/>

JStyle 4.6 is Java source critiquing system. Version 1.0 was introduced to Java developers community in March 1997. Since then it has been continually becoming better - thanks to our customers who have been offering many valuable suggestions to make the product useful in critical Java-based application development.

JStyle can fully analyze the entire JDK 1.3 source (over 1850 files) in under 9 minutes on a Celeron-based machine (466 Mhz) with 128 MB RAM, and in 4 minutes on a PIII (550 Mhz) with 512 MB RAM!

5. Krakatau, Power Software, <http://www.powersoftware.com/>, <http://www.testingfaqs.org/t-static.htm#Krakatau>

Metric Analyzer for Java and C/C++. Krakatau gives metric information on source code at the click of a button.

As well as a comprehensive (over 70 metrics) the user interface gives several graphical methods of sorting and viewing the metric results. This gives quick access to sections of your code with high values for given metrics.

The Project Manager version of the tool can compare versions of your projects and quickly highlight where your code has been changing.

Platforms: Windows NT/2000, Solaris

6. ParaSoft Jtest, Parasoft Corporation, www.parasoft.com, <http://www.testingfaqs.org/t-static.htm#Jtest>

Jtest is a complete automatic class testing tool for Java. Its purpose is to help you increase your Java[tm] software's reliability while dramatically reducing the amount of time you spend testing. Jtest is designed to reduce the burden on Java developers as they write and test their programs from the class level at the earliest stages of development. Jtest automates four essential types of Java testing and gives you the detailed error information you need to make your JavaBeans, APIs and libraries, applications and applets more robust. Jtest automatically performs white-box testing, black-box testing, regression testing, and static analysis (coding standard enforcement) of Java code.

Platforms: Windows NT/95/98/2000, Solaris, Linux

7. Rational Purify, Rational Software Corp, www.rational.com

Error detection. Rational Purify® for UNIX has long been the standard in error detection for Sun, HP, and SGI UNIX platforms. With patented Object Code Insertion technology (OCI), Purify provides the most complete error and memory leak detection available. It checks all application code, including source code, third party libraries, and shared and system libraries. With Purify, you can eliminate problems in all parts of your applications, helping you deliver fast, reliable code.

Free download.

Platforms: Sun, HP, SGI UNIX

3.7.2 C/C++

8. ccount, Joerg Lawrenz, Universitaet Karlsruhe, <http://www.testingfaqs.org/t-static.htm#ccount>

C "readability" analyser. 'ccount' is a set of simple tools for the analysis of the syntactic readability of C source programs. The attributes computed by the ccount tools are things such as length of modules/functions/blocks/ifs/whiles/switches, number of control structures, number of operators in expressions, nesting depth of blocks etc.

Freeware.

Platforms: Most Unix

9. Cleanscape lint-Plus, Cleanscape Software International, www.cleanscape.net/stdprod/lplus/index.html, <http://www.testingfaqs.org/t-static.htm#lint-Plus>

"C" Pre-compile Analysis Tool. A stand-alone source code analyzer that expedites software development by simplifying the debugging and maintenance of "C" programs. lint-Plus detects problems compilers cannot catch, such as "C" global-variable inconsistencies, as well as analyzes source files both individually and as a group.

Platforms: Linux(Alpha, Intel), Unix (Sun, Solaris, HP, IBM, , SGI, Compaq/Digital, Tru64), VMS (VAX, Alpha)

10. CMT++, Testwell Oy, www.testwell.fi, <http://www.testingfaqs.org/t-static.htm#cmt>

CMT++ (Complexity Measures Tool for C/C++) calculates the basic McCabe, Halstead and lines-of-code metrics for C and C++ code. Clear and compact report. Configurable alarm limits. Can measure non-preprocessed source code. Can produce Excel data file for customer specific analysis or for producing graphical representations. Processes 1000s of lines of code in a second and can manage huge code volumes. Does not require a C/C++ compiler, but if Visual C++ is available, CMT++ can be used via its GUI. Read more from <http://www.testwell.fi/cmtdesc.html>

Platforms: Windows 2000/NT/9x, HP-UX, Solaris, Linux

11. CodeWizard, (ParaSoft), <http://www2.parasoft.com/jsp/home.jsp>

C/C++ source code analysis tool. CodeWizard, an advanced C/ C++ source code analysis tool, uses coding guidelines to automatically identify dangerous coding constructs that compilers do not detect. Containing over 240 rules that express industry-respected coding guidelines, CodeWizard allows for the suppression of individual guidelines and includes RuleWizard, a sophisticated tool for customizing these guidelines and creating new ones. CodeWizard is particularly effective on a daily basis at both individual and group levels to simplify code reviews and make code more readable and maintainable.

Platforms: Win NT/2000; Win 95/98/ME; IBM AIX 4.3.x; Compaq Tru64; Unix 5.x, HP-UX 11.x; Linux; HP-UX 10; DEC Alpha 4.x; Sequent 4.x

12. ObjectDetail, ObjectSoftware Inc., www.obsoft.com, <http://www.testingfaqs.org/t-static.htm#objectdetail>

Early defect analyzer and metrics generator tool. ObjectDetail is a software tool to help automate the metrics generation of C++ programs. ObjectDetail analyzes C++ programs to extract class level information. The following reports are generated: encapsulation metric report, inheritance metric report, polymorphism metric report, complexity metric report, miscellaneous metric report.

Platforms: SPARC - SunOs 4.1.X and Solaris 2.X.

13. ParaSoft CodeWizard, Parasoft Corporation, www.parasoft.com, <http://www.testingfaqs.org/t-static.htm#codewizard>

Code standardizing tool. CodeWizard is a unique coding standards enforcement tool that uses patented Source Code Analysis technology (patent #5,860,011) to help developers prevent errors and standardize C++ code automatically. CodeWizard spontaneously enforces C++ coding standards, saving hours of labor-intensive analysis.

CodeWizard Features:

- Enforces over 70 coding standards automatically.
- Pinpoints exact location of programming and design violations.
- Provides a full, detailed explanation for each error found.
- Allows you to edit code directly inside CodeWizard.
- Allows customization of existing coding standards and writing of new coding standards to suit each project or development team.
- Allows flexible suppressions to enforce only the coding standards that are relevant to the current project.

CodeWizard Benefits:

- Reduces bugs by 25%.
- Streamlines development process for quicker time to market.
- Reduces development and support costs.
- Helps improve overall design of code.
- Reduces time spent on maintenance and extension.
- Accelerates learning curve for C++.

Platforms: Windows NT/95/98/2000, Linux, DEC Alpha, IBM RS/6000 (AIX 4.x), HP (HP-UX 10 & 11), SGI (IRIX 6.x), Solaris

**14. PC-lint/FlexeLint, Gimpel Software, <http://www.gimpel.com/>,
<http://www.testingfaqs.org/t-static.htm#pc-lint>**

Static Analysis. PC-lint and FlexeLint will check your C/C++ source code and find bugs, glitches, inconsistencies, non-portable constructs, redundant code, and much more. It looks across multiple modules, and so, enjoys a perspective your compiler does not have.

Platforms: Windows, MS-DOS, OS/2, Unix, Sun, HP, VMS, MVS, VM, OS-9, Mac, etc

**15. Plum Hall SQS, Plum Hall, Inc., www.plumhall.com,
<http://www.testingfaqs.org/t-static.htm#sqs>**

Static Analysis and Code Coverage. For testing C and C++ code, two major components: Guidelines Checker:

Checks source against Plum Hall Programming Guidelines. Terse report highlights fault-prone programming practice for early correction.

CTest++:

Incremental and cumulative code coverage, including correlation of test cases to chunks of code tested. Helps give insight into which code test cases exercise, eliminating unnecessary redundant tests, and speeding development of new tests.

Plum Hall SQS incorporates a parser, not just a lexical analyzer. Flexibly supports test harnesses, whether embedded code, standalone code, manual, or the easy GUI scripting provided, using A-Language.

Platforms: WIN32, (UNIX planned)

**16. PolySpace Verifier, PolySpace Technologies, www.polyspace.com,
<http://www.testingfaqs.org/t-static.htm#polyspace>**

Static Runtime Error Detection Tool. PolySpace Verifier is a tool designed to directly detect run-time errors and non-deterministic constructs at compilation time. Further, PolySpace Verifier does not require execution and modification of the code or time-consuming test cases to be run. Instead, PolySpace Verifier exactly pinpoints the faulty code section that will cause a run-time error if the application was executed.

PolySpace Verifier is available for C and Ada.

Free source code analysis for all visitors.

Platforms: Sun - Solaris 2.6+, Linux - VA-RedHat 6.2.4+

**17. QA C, Programming Research Ltd, www.prqa.co.uk,
<http://www.testingfaqs.org/t-static.htm#PRLQAC>**

Deep-flow static analyser. QA C is a deep-flow static analysis tool which automates your early-cycle testing and inspection processes. A unique feature of the tool is its ability to detect language implementation errors, automatically enforce company or industry programming standards and identify dangerous programming practice. Produces over 45 industry-accepted metrics including Cyclomatic Complexity, Static Path Count and Myer's

Interval and a variety of graphical output reports such as Call Trees, Control Structure and Demographic Analysis.

Platforms: PC (Win NT & 95), Sun (SunOS and Solaris), HP (HPUX), Dec Alpha (OSF1), IBM (AIX), SGI (IRIX), SNI (SINIX)

18. Safer C Toolset, Oakwood Computing Associates Ltd.,

www.oakcomp.co.uk/SoftwareProducts.html,

<http://www.testingfaqs.org/t-static.htm#safer-c>

Static Analysis. The Safer C toolset is to be designed from the beginning using measurement-based feedback. Today, C is more widely used than ever and is the dominant language used in programmable embedded control systems for example. However, the cost of failure in such systems today can be very high indeed. C has many fault and failure modes, but this is balanced by the fact that more is known about how C programs fail than arguably any other language. By avoiding these fault and failure modes, C is capable of producing some of the most reliable systems ever measured, (Hatton (1995)) whilst retaining the many benefits of C such as efficiency, small footprint, portability, availability of experienced engineers and very wide-spread availability of good compilers.

Platforms: Windows, Linux and Sparc Solaris

19. STATIC, Software Research, Inc., www.soft.com/Products/index.html,

<http://www.testingfaqs.org/t-static.htm#STATIC>

Syntac and semantic analysis. STATIC is the static analyzer system for the fully integrated TestWorks/Advisor suite of static source code analyzers and measurement tools. Working as a stand-alone product or as part of the tool suite, STATIC provides more comprehensive syntax and semantic analysis for C programs than most compilers, including locating non-portable constructs and dead code. STATIC also searches the entire program for inconsistencies across the modules that comprise an application. This feature is especially important when analyzing code in multi-programmer projects. STATIC processes a code file or multiple files and generates a report covering more than 300 possible syntactical, warning and informational messages.

Platforms: SPARC SunOS; SPARC; HP-9000

20. STW/Advisor, Software Research, www.soft.com/Products/index.html,

www.methods-tools.com/tools/testing.html

Static analysis suite. The STW/Advisor suite provides static source code analysis and measurement. It comprises METRIC, STATIC, and TDGEN, each of which is also available separately. Seventeen metrics measure a program's data, logic, and size complexity. METRIC is available for C, C++, Ada, and FORTRAN and computes Halstead software science metrics, cyclomatic complexity metrics, and size metrics. STATIC provides syntax and semantic analysis for C programs, including location of dead code and non-portable constructs. TDGEN is a test data generator that takes an existing test script and substitutes new random or sequential data values to create additional tests that more fully exercise the application under test.

Platforms: Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, HP-UX, Digital Unix, Alpha NT

**21. TestBed, Eastern Systems Inc., www.easternsystems.com,
<http://www.testingfaqs.org/t-static.htm#TestBed>**

Static & Dynamic Analysis tool. TestBed is a complete, powerful, integrated tool set that enhances developer's understanding of C, C++, Ada, COBOL, NT, Solaris and Pascal source code and highlights areas of concern for attention. The result is greater software reliability, reduced maintenance costs, and easier re-engineering.

TestBed's Static Analysis capabilities include programming standards verification, structured programming verification, complexity metric measurement, variable cross-reference, unreached code reporting, static control flow tracing, and procedure call analysis. TestBed Dynamic Analysis executes an instrumented version of the source code to detect code coverage defects, statement execution frequency analysis, LCSAJ (Linear Code Sequence and Jump) subpath coverage, and multiple subcondition coverage.

The combination of techniques provides a clear picture of the quality and structure of tested code. In addition, TestBed options such as Tbsafe help users achieve quality certifications including RTCA/DO178B and BS EN ISO9000/TickeIT. TestBed itself was produced under an ISO9000 quality management system of which TestBed is an important part.

3.7.3 Others

**22. Aivosto Project Analyzer, Aivosto Oy, www.aivosto.com/vb.html,
<http://www.testingfaqs.org/t-static.htm#AivostoPA>**

Source code analyzer for Visual Basic. Project Analyzer runs an automated quality check on VB code. It calculates metrics and reports problems related to optimization, coding style and program functionality.

Free evaluation.

Platforms: Windows 95/98/NT/2000

**23. CodeSurfer, GrammaTech, Inc, www.grammatech.com,
<http://www.testingfaqs.org/t-static.htm#CodeSurfer>**

Detailed dependence relations analysing tool. CodeSurfer® is an analysis and inspection tool that lets software developers and maintainers easily understand detailed dependence relations in source code. CodeSurfer provides access to a program's deep structure semantic properties and relationships discovered by global program analyses that conventional tools do not see.

Platforms: Windows, Solaris

24. Dependency Walker, Steve P. Miller, <http://www.dependencywalker.com/>, <http://www.testingfaqs.org/t-static.htm#dependencywalker>

A tool for troubleshooting system errors related to loading and executing modules.

Dependency Walker is a free utility that scans any 32-bit or 64-bit Windows module (exe, dll, ocx, sys, etc.) and builds a hierarchical tree diagram of all dependent modules. For each module found, it lists all the functions that are exported by that module, and which of those functions are actually being called by other modules. Another view displays the minimum set of required files, along with detailed information about each file including a full path to the file, base address, version numbers, machine type, debug information, and more.

Dependency Walker is also very useful for troubleshooting system errors related to loading and executing modules. Dependency Walker detects many common application problems such as missing modules, invalid modules, import/export mismatches, circular dependency errors, mismatched machine types of modules, and module initialization failures.

Freeware.

Platforms: Windows 95, 98, Me, NT 3.51, NT 4.0, and Windows 2000

25. Hindsight/SQA, IntegriSoft, Inc., www.integrisoft.com, <http://www.testingfaqs.org/t-static.htm#hindsight>

Metrics, static analysis. Hindsight/SQA generates metrics from the existing source code.

User metrics can also be accepted for analysis. Metrics can be analyzed and printed out in a variety of report formats.

Platforms: SunOS, Solaris, HP/UX, and IBM AIX

26. McCabe QA, McCabe & Associates, www.mccabe.com, <http://www.testingfaqs.org/t-static.htm#McCabeqa>

Metrics tool. McCabe QA offers:

- * Insight into software quality through module-by-module metric calculation. Metrics including cyclomatic complexity and essential complexity help identify where a program is more likely to contain errors. Metrics measurements are also traced over time to track program improvement.

- * A visual environment for understanding software. Graphical displays represent the structure of code and the metrics rankings to provide valuable assessment of even large systems.

- * A database of software characteristics including metrics and flowgraphs, used as a valuable resource for future software changes and upgrades.

Platforms: Sun - Solaris 2.5.1+, PC - Windows NT 4.0/95, IBM - AIX 4.2+, HP - HP-UX 10.2+, SGI - IRIX 5.3+

27. McCabe Visual Quality Toolset, McCabe and Associates, www.methods-tools.com/tools/testing.html

McCabe Visual Quality Toolset calculates McCabe metrics, Halstead metrics, object-oriented metrics, and other user customizable metrics. It produces structure charts, flow graphs, and metrics reports to display software structures visually. Version: 5.11.
Platforms: Win3.1, Sun Solaris/SunOS, AIX, Silicon Graphics IRIX, DEC VMS, Win 95/98/NT, Digital Unix, HP-UX.

28. METRIC, Software Research, Inc., www.soft.com/Products/index.html, <http://www.testingfaqs.org/t-static.htm#METRIC>

Metric Analysis Tool. METRIC is the software metrics system for the fully integrated TestWorks/Advisor suite of static source code analysers and measurement tools. METRIC works as a stand-alone product or as part of the TestWorks/Advisor tool suite to quantitatively determine source code quality. After processing a source code file, METRIC automatically computes various software measurements. These metrics include the Halstead Software Science metrics, which measure data complexity in routines; the Cyclomatic Complexity metrics, which measure logic complexity in routines; and size metrics, such as number of lines, comments and executable statements.
Platforms: SPARC SunOS; SPARC; HP-9000; DEC-Alpha; NCR 3000; DOS

29. PC-Metric, SET Laboratories, Inc., www.molalla.net/~setlabs, <http://www.testingfaqs.org/t-static.htm#pc-metric>

Software measurement tool. PC-METRIC is a state-of-the-art software measurement tool which performs a variety of static measurements upon a program's source code. PC-METRIC's features do not stop there however. As most software measurement professionals know, the real work starts after the metrics are collected. PC-METRIC supports this task by providing an integrated analysis system that offers a variety of pre-defined reports, graphics and a custom report writing feature. PC-METRIC is the software measurement tool for today's serious software developer. Studies have shown that overly complex code is often less reliable and more difficult to maintain. PC-METRIC assesses the complexity of each function or procedure in an application using a set of industry standard measurement techniques.
Platforms: MS-DOS, UNIX, VMS

30. QA Fortran, Programming Research Ltd, www.prqa.co.uk, <http://www.testingfaqs.org/t-static.htm#PRLQAF>

Deep-flow static analyser. QA Fortran is a deep-flow static analysis tool which automates your early-cycle testing and inspection processes. A unique feature of the tool is its ability to detect language implementation errors, automatically enforce company or industry programming standards and identify dangerous programming practice. Produces industry-accepted metrics including Cyclomatic Complexity, Static Path Count and Myer's Interval

and a variety of graphical output reports such as Call Trees, Control Structure and Demographic Analysis.

Platforms: Sun (SunOS and Solaris), HP (HPUX), Dec Alpha (OSF1), IBM (AIX), SGI (IRIX).

31.SHOWFLOW, GRAY & GRAY Consulting Pty Ltd,
<http://www.testingfaqs.org/t-static.htm#SHOWFLOWDIS>

Development information system (with coverage analysis). SHOWFLOW is a Development Information System for MVS (Multiple Virtual Storage) mainframe. It fully analyses applications written in COBOL, PLI, ASSEMBLER, JCL, REXX, CLIST, CICS, etc and creates a database of all inter-element relationships, along with nice visual flow diagrams to show the flow of control and inclusion relationships - both forwards and backwards. This then allows cumulative knowledge of all the elements that can be flowed to from a given starting point, or conversely, all the start points that can flow to a given element. Static coverage analysis options are included, eg to work out the best (ie minimum) set of test cases which can be used to execute a given set of elements, or list all of the elements that can be executed from a given set of test cases (which for example could then be compared to a list of elements changed for Y2K conversion...). The whole system is fronted by a comprehensive set of easy to use ISPF (Internet Service Providers Forum) panels and menus, including panels to simplify job submission for the analysis and diagramming. Platforms: MVS Mainframe

32.SofAudit, SES Software-Engineering Service GmbH,
www.soring.hu/index.html, <http://www.testingfaqs.org/t-static.htm#SofAudit>

Metrics and quality checks. Supports the evaluation of the legacy systems through metrics and quality checks. SofAudit is a tool for the static analysis of Assembler, COBOL, PL/I and NATURAL programs as well as for IMS, CODASYL, ADABAS and SQL databases. Besides checking the source against predefined rules, it also measures the size, complexity and quality of the programs and data bases. It produces a metric report and a deficiency report.

Platforms: PC Windows (Windows 3.11, Windows95 and Windows/NT)

33.WhiteBox, Reliable Software Technologies, <http://www.cigital.com/>,
<http://www.cigital.com/news/whitebox301.html>

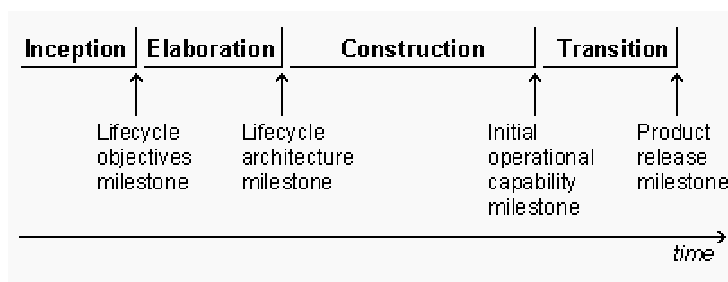
Metrics and code coverage. WhiteBox tool to perform software analysis that is needed to apply a complete WhiteBox methodology solution. The WhiteBox tool collects valuable static code metrics and dynamic code coverage information which are used to assist with test planning for software. Static code metrics provide insight into the complexity of different components of the software; they can be used to help determine where faults are likely to be hiding and therefore where to focus testing resources. Dynamic code coverage is used to determine how much of the code was reached when tests are performed. Using WhiteBox, you can track the code coverage levels obtained for a given test case or a suite of test cases

against the source code to determine missing or incomplete requirements, which when updated can be used to produce additional tests that cause the previously unreachable code to be exercised.

Free evaluation.

RUP ja sen tärkeimmät hyödyt ja tavoitteet

Rational Unified Process (RUP) on työkalu ohjelmistojen suunnitteluun. Se antaa näkökulmia työn suunnitteluun ja suunnittelun vastuuseen. Sen päämääränä on varmistaa korkealaatuinen tuote, joka sopii ennustetun aikataulun ja budjetin sisälle. Ohjelmistojen tulee täyttää käyttäjien vaatimukset, sekä valmistua sovitussa ajassa ja sovitulla rahamäärällä.



Kuva 1: RUP:n vaiheet

Hallinnan kannalta ohjelman elinkaari RUP:ssa on jakautunut neljään ajalliseen vaiheeseen, joista jokainen sisältää oleellisen aikajakson. Vaiheet näkyvät kuvassa 1. Jokaisessa vaiheessa suoritetaan tiettyjä tehtäväkokonaisuuksia.

RUP:lla on kaksi ulottuvuutta:

Horizontaalinen aikaulottuvuus:

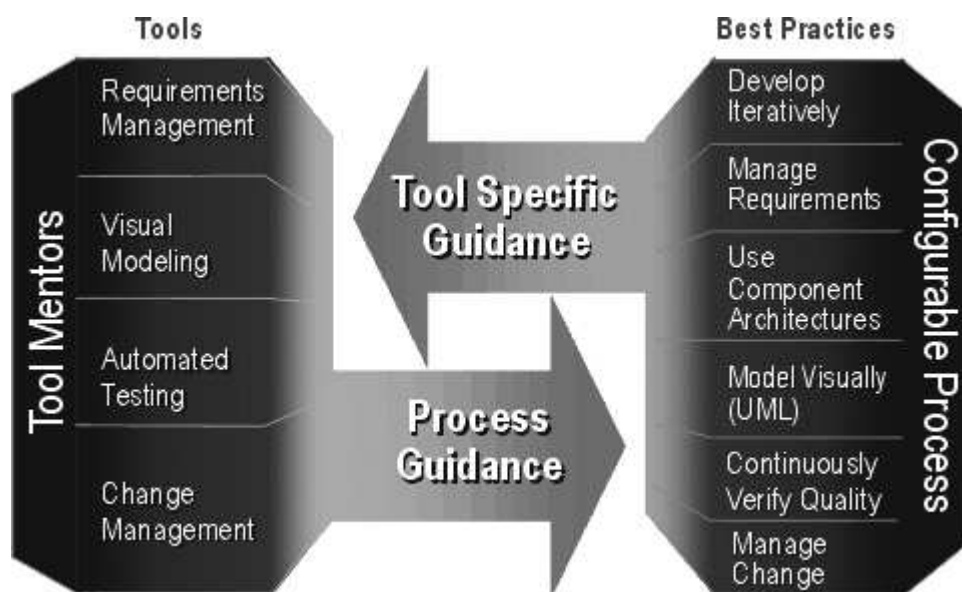
Projektin kannalta hanke on jaettu neljään vaiheeseen: tutkimusvaihe, valmisteluvaihe, rakentamisvaihe ja käyttöönottovaihe. Vaiheet antavat projektille alun, välietapit ja päättävät projektin (voi jatkua toisena projektina).

Vertikaalinen ulottuvuus:

Kuvaa hankkeen etenemistä prosessina vaiheesta toiseen niin, että vaiheet limittyvät ja projektin eri vaiheissa eri tehtävien osuudet vaihtelevat.

RUP:n hyötyihin kuuluu sen mahdollistama tiimin sisäinen kommunikointi, jossa kaikilla jäsenillä on samat työkalut. Tiimin jäsenet ymmärtävät toisiaan, kun kaikilla on sama kieli käytössä, sekä sen luoma mahdollisuus tuottaa laadukkaita ohjelmia. RUP:n avulla saadaan juuri se tieto, jota kullakin hetkellä tarvitaan. RUP auttaa myös UML:n käytössä sekä ohjaa ohjelmoinnin piirteissä. RUP näyttää miten voidaan lisätä parhaat toiminnot ohjelmistosuunnitteluun, ja kuinka työkaluja voidaan käyttää automatisoimaan suunnittelua.

Best Practices ja niiden tarkoitus



Kuva 2: Best Practises

Best Practices (parhaat käytännöt) sisältää kuvassa 2 näkyvät asiat. Best Practicesin välineet ovat vaatimustenhallinta, visuaalinen mallintaminen, automaattinen testaus ja muutostenhallinta.

Ensimmäisenä on *iterative development* (iteratiivinen kehitys), jossa elinkaari koostuu useasta iteraatiosta. Tällaisessa kehitystavassa tehdään eri vaiheita (esim. liiketoiminnan mallinnus, vaatimusmäärittely, analyysi ja suunnittelu) useaan kertaan. Jokaisella iteraatiokierroksella parannetaan ja/tai tarkennetaan edellisen vaiheen tuloksia. Iteratiivisen kehityksen tarkoituksena on minimoida riskejä, jotka ilmenevät usein vasta kehityksen loppuvaiheessa. Mitä aikaisemmassa vaiheessa virhe löydetään, sitä edullisempaa ja vähemmän aikaa vievää virheen paikantaminen ja korjaaminen yleensä on.

Requirements management (vaatimustenhallinta) on systemaattinen lähestymistapa löytää, dokumentoida, organisoida sekä jäljittää vaatimusten muutokset. Avain tehokkaaseen vaatimusten hallintaan sisältää selvän kertomuksen vaatimuksista, niiden attribuuteista ja jäljitettävyydestä muihin vaatimuksiin sekä muiden projektien toiminnot. Vaatimusten hallintaa käytetään, jotta varmistuttaisiin siitä, että asiakkaalla ja valmistajalla on yhteinen näkemys käytetyistä käsitteistä ja vaatimuksista.

Component Architecture (komponenttiarkkitehtuurit) sisältää komponentit, jotka ovat yhtenäinen ryhmä koodia, lähdekoodi tai toteuttamiskelpoinen koodi, jossa on tunnistetut rajapinnat ja toiminnot, jotka antavat tiedon niiden sisällöstä ja ovat siten korvattavissa. Komponentteihin perustuva arkkitehtuuri vähentää ohjelmiston monimutkaisuutta ja ohjelmat siten ovat vakaampia.

Visual modeling (visuaalinen mallintaminen) auttaa käyttämään semanttisuutta oikein, graafisesti ja tekstillisesti suunnittelemaan notaation ohjelmiston suunnittelussa. Notatio, kuten UML, sallii abstraktin tason nousta samalla kun pitää yllä tarkkaa syntaksia ja semantiikkaa. Tämä parantaa suunnitteluryhmän kommunikointia ja mallin esittämistä sekä systeemin toiminnallisuutta. Malli (model) on yksinkertainen näkymä systeemistä, kuten Use Case, luokkadiagrammi jne.

Continuously Verify Quality (jatkuva laadunvalvonta) perustuu siihen, että laatua tulisi tarkastaa ja testata jokaisen iteraation jälkeen. Komponenttien valmistuttua suoritetaan integrointitestaus muiden komponenttien välillä, eikä vasta projektin loppuvaiheessa, kuten perinteisissä menetelmissä.

Manage Change (muutostenhallinta) on enemmän kuin vain tarkastus sisältä ja ulkoa. Se sisältää työympäristön hallinnan, kehityksen seurannan, integroinnin ja rakentamisen. Haastavaa ohjelman kehityksessä on, että joudutaan käsittelemään monia kehittämiä, organisoimaan erilaisia tiimejä, työskentelemään yhdessä erilaisten iteraatioiden, yhteyksien, tuotteiden ja mallien kanssa. Tästä saattaa tulla kaaos. RUP:ssa annetaan malli sille, miten nämä saadaan koordinoitua yhteen. Dokumentoinnin ja muutostenhallinnan tavoitteena on tehostaa kehitysprosessia ja minimoida valmiin ohjelman virheiden määrä.

RUP:n kehittämisprosessin vaiheet sekä vaiheiden tehtävät ja tulokset

RUP:n kehittämisprosessin vaiheet ovat tutkimus-, valmistelu-, suunnittelu-, rakennus- ja käyttöönottovaihe.

Tutkimusvaihe (inception) tehtävät:

- Todeta projektin ohjelmiston laajuus ja olosuhderajoitteet sisältäen toiminnallisen vision, kriteerien hyväksymisen ja sen millainen tuotteen tulisi olla ja millainen ei.
- Kuvata systeemin kriittiset käyttötapaukset, tulevaisuudenkuvat operaatioista.
- Esittely tai demonstraatio ainakin yhdestä arkkitehtuurikandidaatista verraten johonkin pääasialliseen tulevaisuuden kuvaan.
- Kustannusten arviointi ja koko projektin aikatauluttaminen
- Riskien arviointi
- Kannustavan ilmapiirin luominen projektille

Ensimmäisen vaiheen tuloksena saadaan päätös projektin jatkamisesta tai hylkäämisestä ja sopimus kaikkien asianosaisten kanssa projektin elinkaaren tavoitteista.

Valmisteluvaiheen (elaboration) tehtävät:

- Varmistaa, että arkkitehtuuri, vaatimukset ja suunnitelmat ovat tarpeeksi vakaita ja riskit riittävästi lievennettyjä, jotta pystytään ennustettavasti arvioimaan kehittämisen täydentämisen aikataulu ja kustannukset.
- Osoittaa kaikki arkkitehtuurisesti merkittävät riskit projektissa
- Tuottaa kehityskelpoinen tuote-laatu-komponenttien prototyyppi
- Demonstroida, että arkkitehtuuri tukee systeemin vaatimuksia järkevällä hinnalla ja järkevässä ajassa

Toisen vaiheen tuloksena saadaan arkkitehtuurin elinkaari

Suunnitteluvaiheessa (planning) laaditaan suunnitelma rakennusvaiheen läpiviemisiksi

- Jaetaan työt käyttötapausten perusteella siten, että jokaiselle käyttötapauskelle määritellään toteutusvaihe
- luokitellaan käyttötapaukset ja määritellään niille prioriteetti sekä tarvittavat taloudelliset ja ajalliset resurssit
- arvioidaan riskejä jokaisen käyttötapausten kohdalla

Rakentamisvaihe (construction) toteutetaan läpikäymällä iteraatiokierroksia. Tehtävät:

- Minimoida kehityskustannukset optimoimalla resurssit ja välttämällä tarpeetonta hylkytavaraa ja uudelleen tekemistä
- Saavuttaa riittävä laatu niin nopeasti kuin on käytännöllistä
- Suorittaa analyysia, suunnittelua, kehitystä ja testausta kaikilla vaadituilla toiminnoilla
- Kehittää täydellinen tuote, joka on valmis siirtymään käyttäjilleen. Tämä merkitsee mm. käyttötapausten ja muiden vaatimusten kuvaamista ja toteutusta, sekä testaamista.
- Suunnitelma rakennusvaiheen läpiviemisestä käyttötapausten avulla

Kolmannen vaiheen jälkeen voidaan määrittää, onko tuote valmis siirtymään testiympäristöön

Käyttöönotto vaihe (transition) tarkoituksena on varmistua, että sovellus on loppukäyttäjän käytettävissä. Tehtävät riippuvat tavoitteista. Tämän vaiheen lopussa elinkaaren tehtävät pitäisi olla läpikäyty ja projektin pitäisi olla siinä tilassa, että se voidaan päättää. Tehtäviä:

- testataan vastaako tuote asetettuja vaatimuksia
- testataan ovatko rinnakkaiset toiminnot vastaavia kuin systeemissä, jonka se korvaa
- toiminnallisten tietokantojen konvertointi
- markkinointi, jakaminen, myynti
- kaupalliset paketit ja tuotanto, laaja henkilöstökoulutus
- "virittävät" toiminnot, kuten virheiden korjaaminen, suorituskyvyn parantaminen ja käytettävyys.
- tuotekriteerien hyväksyminen
- saavuttaa käyttäjien hyväksyntä
- asianosaisten sopimuksen saavuttaminen siitä että sijoituksen perusviivat ovat täydellisiä ja yhtenäiset vision kriteeriärvion kanssa

RUP:n ja UML:n yhteistyö

RUP käyttää mallintamisessa hyväkseen UML (Unified Modeling Language)- notatiota. UML antaa yksinkertaistetun näkymän systeemiin näyttäen systeemin olennaiset osat tietystä näkökulmasta ja piilottaen ei- olennaiset yksityiskohdat.

Notaation mukaiset kaaviot toimivat yhteisenä kommunikointivälineenä ohjelmistokehitysprosessin eri toimijoiden välillä. UML tarjoaa semantiikan, joka on tarpeeksi monipuolinen esittämään kaikki tärkeät strategiat ja taktiset päätökset.

UML mahdollistaa tarkan ja selkeän mallin esittämisen vaatimukset, jolloin eri asianosaiset voivat tarkastella vaatimuksia luotettavasti. Lisäksi UML mahdollistaa eri vaihtoehtojen löytämisen ja vertailemisen pienillä kustannuksilla. UML luo myös pohjan implementoinnille.

AnalystStudion tuotteet sekä niiden käyttötarkoitukset ja tavoitteet?

Julkaisuhuomautukset (Release Notes):

Julkaisuhuomautukset (Release Notes) kuuluvat jokaiseen Rational -tuotteeseen. Tavoitteena on, että ne antavat tietoja, jotka ovat erittäin tärkeitä asennuksessa ja Rational -tuotteiden käytössä.

Rational ClearCase LT:

Väline ohjelmistokehitysprosessin muutostenhallintaan. Sen avulla voidaan tehokkaasti hallita eri vaiheiden muutoksia ja tuotoksia, kuten vaatimuksia, visuaalista mallintamista, dokumentaatiota ja testausta.

Rational ClearQuest:

Virheiden seuranta ja muutospyyntöjen hallintatyökalu. Rational ClearQuest on joustava työkalu, jolle eri laitealustat, tiimien monimutkaisuus tai maantieteellinen sijainti eivät tuota ongelmia.

Rational RequisitePro:

Optimoi vaatimustenhallintaa. On helppokäyttöinen työkalu vaatimustenhallintaan, joka aikataulun ja kustannusten puitteissa auttaa luomaan sovelluksen, johon asiakas on tyytyväinen.

Rational Rose:

Tarjoaa ohjelmistokehitykseen visuaaliset mallinnusvälineet. Perustuu UML:ään.

Rational SoDA:

Työkalu raporttien generointiin. Tukee päivittäistä raportointia ja dokumentaatiota. SoDA:n tarkoituksena kerätä tietoa eri lähteistä ja tehdä valitun mallin mukainen raportti.

Rational TestManager:

Väline ohjelmiston testauksen hallintaan. Sen avulla jokainen kehitysprosessiin osallistuja voi tarkastella testausta ja sen tuloksia omasta perspektiivistään lähinnä valvomalla, että testaus on suoritettu kattavasti.

Rational Unified Process:

Ohjelmistokehitysprosessin mallinnusväline. Sen avulla voidaan määritellä kunkin eri vaiheen ja projektiin osallistuvan henkilön tehtävät ja vastuut. Tavoitteena on tuottaa ennalta sovitun aikataulun ja budjetin puitteissa korkealaatuisia ohjelmistotuotteita, jotka vastaavat asiakkaan tarpeita ja vaatimuksia.

Rational Administrator:

Tavoitteena on keskittää projektin hallintaa ja hallita Rational- tuotteiden tietovarastojen välisiä suhteita, sekä Rational Testin käyttäjiä ja käyttäjäryhmiä. Administratoria käytetään projektin luomiseen, yhdistämiseen ja testitietokannan luomiseen, yhteyden muodostamiseen testituotteiden ja Requisite Pro:n välille, sekä Rational Clear-Quest- tietokannan luomiseen.