UDK 004.412
Survey paper

FABASOFT BEST PRACTICES AND TEST METRICS MODEL

Nadica Hrgarek
Fabasoft R&D Software GmbH & Co KG, Linz, Austria
Nadica.Hrgarek@fabasoft.com

Abstract: *Software companies have to face serious problems about how to measure the progress of test activities and quality of software products in order to estimate test completion criteria, and if the shipment milestone will be reached on time. Measurement is a key activity in testing life cycle and requires established, managed and well documented test process, defined software quality attributes, quantitative measures, and using of test management and bug tracking tools. Test metrics are a subset of software metrics (product metrics, process metrics) and enable the measurement and quality improvement of test process and/or software product. The goal of this paper is to briefly present Fabasoft best practices and lessons learned during functional and system testing of big complex software products, and to describe a simple test metrics model applied to the software test process with the purpose to better control software projects, measure and increase software quality.*

Keywords: *software metrics, software quality, software test, test metrics.*

1. INTRODUCTION

People are not perfect, they make mistakes during development in design and code, and software is developed by people. Software defects (bugs, failures) play more and more important role in our everyday life, and cost a lot of money (e.g. Ariane 5 explosion; loss of Mars climate orbiter).

Development organizations who deliver software–based systems have to face serious problems about how to control the progress of test activities and quality of software products throughout the project life cycle in order to estimate test completion criteria, and if the shipment milestone will be reached on time. Software is becoming more complex, bigger and unsafe in safety-critical systems (e.g. computer controlled radiation therapy machine Therac-25 overdosed six people; the US Vicennes shot down the Iranian Airbus 320, which was mistaken for an F-14 and 290 human lives lost; Patriot missile hit an American military barracks). So in summary, managing software quality is necessary to deliver high quality, and trustworthy (reliable, secure and safe) software products.

Measurement is a key activity in testing life cycle and requires established, managed and well documented test process, defined software quality attributes, quantitative measures, and using of test management and bug tracking tools.

Testing is the process of executing a program or system with the intent of finding errors. [7, p. 4] The goal of testing activities is to reduce risk and uncover as many faults (bugs, defects) in software as possible. Of the one part testing cannot guarantee the correctness of software but of the other part can be effectively used to find defects.

Many software organizations spend in average 30 to 50 percent of their software budget on testing, but many software projects fail or delivered software is frequently unreliable because of quality problems. Good testing should uncover serious quality problems so called showstoppers. Defects detected too late in software life cycle lead to large amounts of additional defect removal costs.

Software testing is not only critical factor to improve software quality, it is also very important to improve software (development) process itself. This paper presents the best software quality assurance practices and simple test metrics model that contribute to improved software testing process and software quality. The list of the Fabasoft software engineering and quality assurance best practices, and test metrics is primarily focused on software testing process.

2. FABASOFT QUALITY ASSURANCE BEST PRACTICES

Fabasoft is a leading manufacturer of standard software for electronic government and enterprise content and records management. Fabasoft target customer segment is large-scale service organizations both in the public sector (Fabasoft eGov-Suite[1]) and in the private sector (Fabasoft eCRM-Suite[2]).

The importance of quality assurance and its impact on software at Fabasoft is not underestimated. Quality assurance is an integral part of the software development process and maintenance. Software quality must be built in from the beginning and every Fabasoft software professional/company employee is responsible to ensure that his or her work is correct. Software quality engineers are integrated in the testing team and must ensure that software developers are doing high quality work. Finding undiscovered defects in the development stage results in easier correction, improves customer satisfaction, and is more cost-effective. In addition, software quality assurance must be applied to the software process itself (compliance with ISO 9001:2000 standard).

In order to ensure the delivery of high quality software products, Fabasoft ensures software quality assurance during the whole software development life cycle, and uses the following software engineering and quality improvement best practices.

2.1. USE CASE SPECIFICATIONS

Fabasoft software development process is based on the new German model V-Modell XT (extreme tailoring) for planning and realizing (software) projects, unified process (UP) model, and advanced use case modelling concept. Use cases are used as basis for user documentation, and deriving test cases in test plans. They help identify features to be tested and help design the required test cases.

Use case specifications are very helpful for software quality engineers because they provide necessary information about input data and expected output results. Without good written specifications software quality engineer can not effective perform tests.

[1] Integrated and public sector certified product for document management, workflow, file and process management, and content management with authors' Web portal.

[2] Integrated customer relations management, and enterprise content management for private service organizations.

2.2. CODE REVIEWS

A source code (peer) review is a part of static testing and provides security, reliability and functionality wins. The purpose of code reviews is to detect product defects before they are passed on to another software development phase or released to the customer.

In order to provide the best value for improving software quality, code reviews are usually performed by Fabasoft skilled development project managers.

Number of defects detected by code reviews is proposed measure for counting the defects in the source code program. To determine code review process effectiveness, for each code review is recommended to collect and track necessary effort which contains the total hours spent on various part of the code review event (preparation, meeting, rework, etc.).

Software reviews and audits are detailed described in IEEE standard 1028:1997 (for details see [2]).

2.3. UNIT TESTING

Unit or component testing is the testing of individual software components. [3] Unit testing tests units (e.g. objects, classes, methods, functions, procedures) in isolation (ideally no interaction with other components).

Fabasoft software engineers are primary responsible to specify and perform (automated) unit tests to make sure most faults are found during unit testing. To perform unit testing, they use Rational Purify and Rational Quantify tool.

For more information about software unit testing see IEEE standard 1008:1987.

2.4. FUNCTIONAL AND NON-FUNCTIONAL TESTING

Functional testing is based on test inputs which are generated using program specifications. It tests how well software meets the functionality requirements. To achieve the best testing results is necessary to perform as functional as non-functional tests. Functional tests are performed manually and automated as far as possible (Fabasoft UCQ).

Non-functional tests are carried out manually and automated in Abilities Lab department. Non-functional testing includes: interface test, setup test, stress test, documentation test, configuration test, performance test, and load test.

2.5. MULTI-PLATFORM TESTING

Fabasoft software products and the Fabasoft reference architecture are available for both Microsoft Windows and Linux environments to provide optimal integration into existing infrastructures.

Fabasoft software products are tested and available with the same high quality on both operating systems platforms in combination with different web browsers and groupware.

2.6. DAILY AND WEEKLY BUILDS

Daily builds is a well known software engineering best practice. Every build contains software release with changes that are being promoted into the change control system.
The advantage of using daily builds is that the newer releases of software are every day available to developers and testers. Regression tests after building discover integration problems where a change breaks the build.

2.7. AUTOMATED TEST EXECUTION WITH FABASOFT UCQ TOOL

Fabasoft UCQ (Use Case Quality Automation) tool supports use case based and automated software quality assurance for Fabasoft products and project solutions. The tool employs predefined use cases for the automated testing of system environments during system modifications or version upgrades. The recorded use cases are automatically executed in order to verify whether the installation was properly performed. Any errors are logged, displayed, and remedied in a timely and targeted fashion. Use of Fabasoft UCQ is especially helpful for regression testing.

2.8. RISK BASED TESTING WITH SMOKE TESTS

Smoke tests are coarse form of regression test to determine that the software product doesn't crash as a result of recent changes.

Smoke test is usually applied to daily build to see if there is any „smoke" which is sign of new errors.

To perform smoke testing Fabasoft has implemented priority for each test case in test plan. Test cases with the highest priority and risk are always first performed as smoke tests to detect major problems of main functionalities in product. The supported functionalities have less priority and aren't performed with smoke tests.

2.9. DEFECT TRACKING AND RESOURCE MANAGEMENT

Testing is a group of activities that can be planned in advance, managed and performed systematically. Most of the defects are being detected during different phases of testing.

Resource and defect management helps to understand the dynamics of software development and test process. It is necessary to measure and predict product quality, to predict shipment milestone, to estimate test completion criteria, etc. Defect data can be used in project tracking and analysis and help manager to evaluate project progress and improve project planning.

Every defect (quality problem, incident, deviation, issue) and requirement in Fabasoft software products and/or projects is documented / recorded as implementation order (Figure 3) by ID, priority and severity (0 – critical with the highest priority, A – major with high priority, B, C, D – minor with the lowest priority), and problem type (e.g. total failure, defect, loss of data, security, performance, project management, documentation, new functionality, etc.).

Quality problems and requirements are reported back to the software engineer, describing the wrong and expected behaviour.

Fabasoft internal quality assurance and product planning tool is knowledge base for requirements and quality problems, and also has the ability to visualize test milestones and programmer's work load.

3. SOFTWARE METRICS

Much attention has been focused recently on quality assurance as well as software measurement. Software measurement is a continuous process that has to be integrated into the software development process. This process supports the management goals to have predictable outputs, assess project status, reduce risks, early detect problem areas, increase product/process quality, and customer satisfaction.

Kan [5] classifies metrics into the following three categories: product metrics, process metrics, and project metrics (see Figure 1).

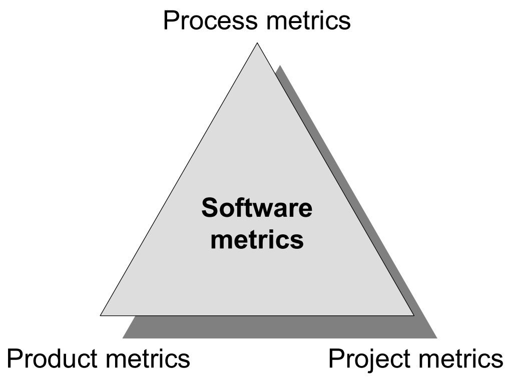


Figure 1. Software metrics classification

Software quality metrics are a subset of software metrics that focus on quality aspects of the product, process, and project. [5, p. 85]

Software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality. [4, p. 3]

Many software metrics have been defined: lines of code (LOC), McCabe's cyclomatic complexity (CC), defect density (DD), Albrecht's function points (FP), failure rate (FR), expected (estimated) number of failures, mean time to failure (MTTF), defect removal efficiency (DRE), memory size, defect severity, rework effort, comment percentage (CP), Chidamber & Kemerer object oriented metrics suite [1], MOOD metrics set, etc.

4. TEST METRICS MODEL

The objective of testing is to have the highest likelihood of finding the most errors with a minimum amount of timing and effort.

In order to manage and control the software testing process, it has to be measured. Similarly to Kan [5], Konda [6, p. 36] classifies test metrics into three categories: product, project and process. Test metrics are a subset of software metrics (product metrics, process metrics) and can be used to measure and improve quality of test process and/or software product.

Collecting test metrics is not easy activity, but is a necessary aspect of software testing process. Currently more than 10 metrics are being collected in this study including rework effort, actual defect counts and measures of test coverage. The collected test metrics are used as a basis for software quality measurement and reporting.

This paper intends to propose a simple test metrics model, experimented in several completed Fabasoft quality assurance projects.

Test metrics modl represents an optimum set of black-box test metrics at the system test level to improve the software quality. In the following some of the best known test metrics are described.

4.1. TEST CASES COVERAGE (TCC)

The basic component of testing is a test case which describes inputs, expected and actual result. Black-box test cases set is organized in Fabasoft internal quality assurance tool into test plan.

Test cases coverage is a simple derived metric for measuring test coverage which shows the relation between executed test cases and total number of specified test cases. TCC measure ensures that all test cases have been executed at least once.

A very important question in software testing is how do we know when testing is complete. TCC provides the information about the testing activities progress and test completion criteria, and is used to assess the scope of the testing process.

$$TCC = \frac{test\ cases\ executed}{total\ test\ cases} \qquad (1)$$

4.2. NUMBER OF TEST CASES FAILED (TCF) AND PASSED (TCP)

If the actual result after test case execution varies from the expected result, then a quality problem has been detected. In one test case zero, one or more quality problems can be found.

Every test case that contains one or more quality problems / requirements is tracked as failed test case. The more test cases that fail, the more effort required to correct the quality problem.

$$TCF = test\ cases\ executed - passed\ test\ cases \qquad (2)$$

$$TCP = test\ cases\ executed - test\ cases\ failed \qquad (3)$$

4.3. NUMBER OF TEST CASES CLARIFIED (CTC)

This is a measure of the total number of test cases that need to be clarified. The more test cases that need to be clarified, the more wasted test iterations occurs.

4.4. NUMBER OF TEST CASES RUNS (TCR)

For every test case in test plan is possible to get the number of test cases runs. This measure is especially important for regression testing.

4.5. NUMBER OF DETECTED DEFECTS (NDD)

One of the most popular and very used test metrics is the number of detected defects (faults, quality problems) in software per reporting time period.

Quality problems indicate errors or defects in the existing functionalities of the Fabasoft software products i.e. projects. Quality problems are recorded in Fabasoft internal quality assurance tool as implementation requests (Figure 2).

Number of critical problems so called showstoppers is useful to know before and after the shipment. All known showstoppers must be repaired before the delivery.

4.6. DEFECT DENSITY (DD)

Defect density shows relationship between number of discovered defects and product size (e.g. KLOC). Large number of reported defects (DD>25%) can be result of requirements not being met, inadequate testing, or poor code quality.

In our test metrics model we use DD as relationship between TCF and executed test cases.

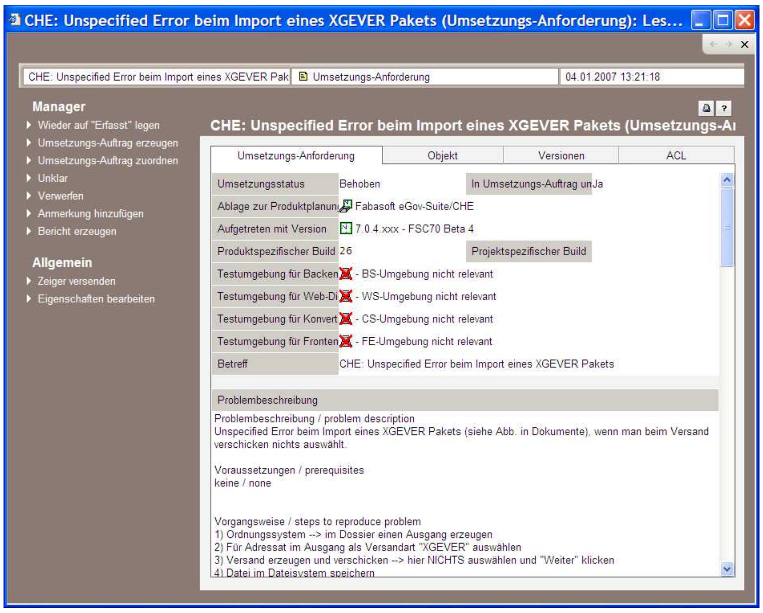
$$DD = \frac{number\ of\ defects}{product\ size} \tag{4}$$

$$DD = \frac{TCF}{test\ cases\ executed} \tag{5}$$

4.7. NUMBER OF IMPLEMENTATION REQUESTS (NIR)

All (user) requirements, problems and quality problems are collated in Fabasoft internal quality assurance and product planning system as new implementation requests (Figure 2) with an implementation status (new, closed, solved, open, rejected, etc.).

Implementation request elements are: implementation status, project, version, build, summary, problem description, attachment, problem type, priority in project, to be version, to be milestone, use cases, etc.


CHE: Unspecified Error beim Import eines XGEVER Pakets (Umsetzungs-Anforderung): Les...
CHE: Unspecified Error beim Import eines XGEVER Pak
Umsetzungs-Anforderung
04.01.2007 13:21:18
Manager
Wieder auf "Erfasst" legen
Umsetzungs-Auftrag erzeugen
Umsetzungs-Auftrag zuordnen
Unklar
Verwerfen
Anmerkung hinzufügen
Bericht erzeugen
Allgemein
Zeiger versenden
Eigenschaften bearbeiten
CHE: Unspecified Error beim Import eines XGEVER Pakets (Umsetzungs-A
Umsetzungs-Anforderung
Objekt
Versionen
ACL

Umsetzungsstatus	Behoben	In Umsetzungs-Auftrag un	Ja
Ablage zur Produktplanun	Fabasoft eGov-Suite/CHE		
Aufgetreten mit Version	7.0.4.xxx - FSC70 Beta 4		
Produktspezifischer Build	26	Projektspezifischer Build	
Testumgebung für Backen	- BS-Umgebung nicht relevant		
Testumgebung für Web-Di	- WS-Umgebung nicht relevant		
Testumgebung für Konvert	- CS-Umgebung nicht relevant		
Testumgebung für Fronten	- FE-Umgebung nicht relevant		
Betreff	CHE: Unspecified Error beim Import eines XGEVER Pakets		

Problembeschreibung

Problembeschreibung / problem description
Unspecified Error beim Import eines XGEVER Pakets (siehe Abb. in Dokumente), wenn man beim Versand verschicken nichts auswählt.

Voraussetzungen / prerequisites
keine / none

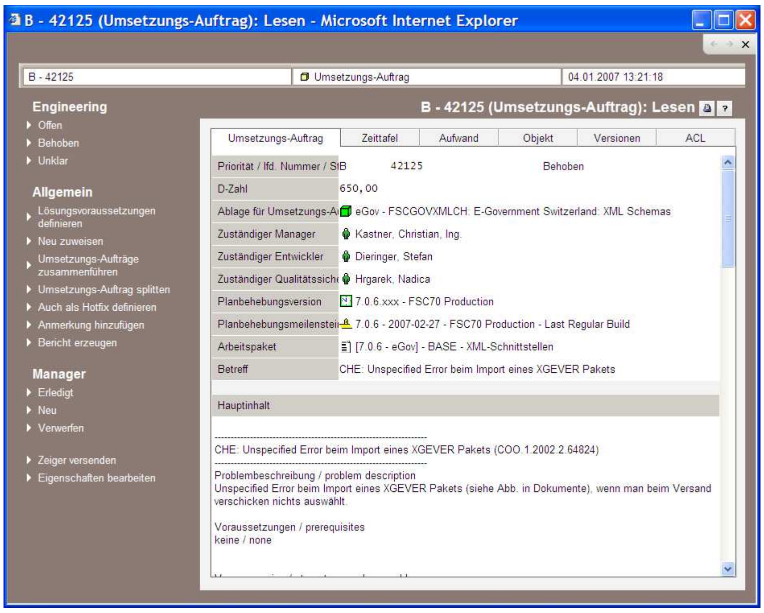
Vorgangsweise / steps to reproduce problem
1) Ordnungssystem --> im Dossier einen Ausgang erzeugen
2) Für Adressat im Ausgang als Versandart "XGEVER" auswählen
3) Versand erzeugen und verschicken --> hier NICHTS auswählen und "Weiter" klicken
4) Datei im Dateisystem speichern

Figure 2. Sample implementation request

4.8. NUMBER OF IMPLEMENTATION ORDERS (NIO)

Implementation order (Figure 3) is a track of quality problem or requirement. The head of the product development project assesses the quality problems or requirements described in implementation request and create implementation order.

Implementation order elements are: ID (priority "-" number), responsible software engineer (developer), software quality engineer, and manager, (planned) fixing version, implementation status, implementation request (1:n), component folder, fixing build, summary, problem type, problem change log, open date, close date, etc.



B - 42125 (Umsetzungs-Auftrag): Lesen - Microsoft Internet Explorer

B - 42125 | Umsetzungs-Auftrag | 04.01.2007 13:21:18

B - 42125 (Umsetzungs-Auftrag): Lesen

Priorität / lfd. Nummer / StB	42125	Behoben
D-Zahl	650,00	
Ablage für Umsetzungs-A	eGov - FSCGOVXMLCH: E-Government Switzerland: XML Schemas	
Zuständiger Manager	Kastner, Christian, Ing.	
Zuständiger Entwickler	Dieringer, Stefan	
Zuständiger Qualitätssiche	Hrgarek, Nadica	
Planbehebungsversion	7.0.6.xxx - FSC70 Production	
Planbehebungsmeilenstei	7.0.6 - 2007-02-27 - FSC70 Production - Last Regular Build	
Arbeitspaket	[7.0.6 - eGov] - BASE - XML-Schnittstellen	
Betreff	CHE: Unspecified Error beim Import eines XGEVER Pakets	

Hauptinhalt

```
------------------------------------------------------------------
CHE: Unspecified Error beim Import eines XGEVER Pakets (COO.1.2002.2.64824)
------------------------------------------------------------------
Problembeschreibung / problem description
Unspecified Error beim Import eines XGEVER Pakets (siehe Abb. in Dokumente), wenn man beim Versand
verschicken nichts auswählt.

Voraussetzungen / prerequisites
keine / none
```

Figure 3. Sample implementation order

4.9. IMPLEMENTATION REQUESTS DISTRIBUTION

To easy identify quality problems using tables and charts (e.g. histograms, pie charts), Fabasoft uses the implementation requests distribution by problem type (see Figure 8), component, product planning folder, and implementation status (see Figure 7).

4.10. NUMBER OF REMAINED DEFECTS IN PRODUCTION VERSION (DIP)

Only exhaustive testing can show a program is free from defects. Even simple programs demonstrate that exhaustive testing is impossible in practice. Exhaustive testing to find every fault in software system is impossible because of too many possible paths, inputs and user environments. Bugs will be found by customers after the product is released.

Number of unfixed defects released to production is a metric of production release quality. DIP measure is the number known defects after shipment and will be corrected in a new release.

$$DIP = \text{total number of release defects} - \text{number of corrected release defects} \quad (6)$$

4.11. REWORK EFFORT (RE)

Rework effort is recorded for every implementation order (1 hour is equivalent to 0,125) as the required time to solve and close a quality problem / requirement.

This metric provide information about passed as well as to predict future rework effort to fix a quality problem or to implement a (user) requirement.

4.12. TOTAL STAFF LEVEL

One simple measure that can be collected in software projects is staff (personnel) level, which counts the total number of software personnel available for a project.

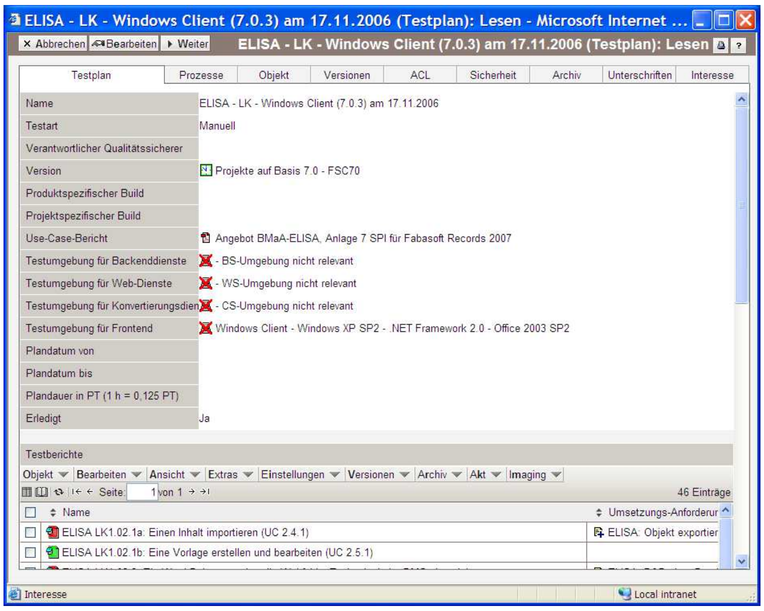
The results can give software project managers a coarse indication of whether they will have enough personnel for a project or whether they will have to start looking for new team members.

5. TEST METRICS MODEL CASE STUDY

Before starting to write this paper, an agreement is made with the company to outline the boundaries of the study. It is decided that the actual data will not been shown in tables and presented on the charts. Before drawing charts, the data will be multiplied by a constant factor.

The focus of this paper is on the development and testing of the Fabasoft Folio components. These components were developed by Fabasoft for the ELISA project. Many test scenarios and test cases were executed during multiple test activities by two software quality engineers. Metrics based on test data collected and/or calculated from test plan (see Figure 4) which contains test cases, implementation requests and orders at the system test level are summarized in Table 1. A metrics collection program was initiated with a small set of test metrics. Test data were collected from two test plans of a real-life Fabasoft software project, build 27 and 29. The reason for doing this is to measure the improvements in the test process and software project.

Analyzing and using measurement test data (see Table 1) can allow managers to have objective information which helps them to make the decisions necessary to run their projects. Collecting this low level, simple test data also helps them by providing the basis for possibly improved estimates for future projects. It also allows project/product managers to use it in a variety of different analyses. Test metrics help test managers to get insight in running test projects.



Name	ELISA - LK - Windows Client (7.0.3) am 17.11.2006
Testart	Manuell
Verantwortlicher Qualitätssicherer	
Version	Projekte auf Basis 7.0 - FSC70
Produktspezifischer Build	
Projektspezifischer Build	
Use-Case-Bericht	Angebot BMaA-ELISA, Anlage 7 SPI für Fabasoft Records 2007
Testumgebung für Backenddienste	- BS-Umgebung nicht relevant
Testumgebung für Web-Dienste	- WS-Umgebung nicht relevant
Testumgebung für Konvertierungsdien	- CS-Umgebung nicht relevant
Testumgebung für Frontend	Windows Client - Windows XP SP2 - .NET Framework 2.0 - Office 2003 SP2
Plandatum von	
Plandatum bis	
Plandauer in PT (1 h = 0,125 PT)	
Erledigt	Ja

Name	Umsetzungs-Anforderur
ELISA LK1.02.1a: Einen Inhalt importieren (UC 2.4.1)	ELISA: Objekt exportier
ELISA LK1.02.1b: Eine Vorlage erstellen und bearbeiten (UC 2.5.1)	

Figure 4. Sample test plan for an Fabasoft project

Table 1. Test data

Project: ELISA Version: 7.0.3 Client: .NET Operation system: Windows XP Professional SP2 Web browser: Microsoft Internet Explorer 6.0 SP2		
Test date	13.11.2006	17.11.2006
Test build ID	27	29
Total test cases defined	46	46
Test cases executed	44	41
Test cases not executed	2	5
Test cases clarified (CTC)	2	2
Test cases coverage (TCC) (%)	95,65	89,13
Test cases passed (TCP)	21	30
Test cases failed (TCF)	23	11
Defect density (DD) (%)	52,27	26,83
Total implementation requests (NIR)	32	14
Total implementation orders (NIO)	19	3
Total defects released to production (DIP)	6	9
Number of software quality engineers (testers)	2	1
Number of software developers	1	1
Number of project/product managers	2	2
Total staff level	5	4
Total test cases executed by tester A	17	41

Total test cases executed by tester B	27	0
Implementation requests by implementation status		
New	6	9
Closed	19	0
Solved	0	0
Open	0	0
Rejected	7	6
Unclear	0	0
Delayed	0	0
Pending	0	0
Implementation requests by problem type		
Failure	19	7
New functionality	2	3
Defect	3	1
Optical / usage	2	1
Documentation	0	0
Crytical with message	5	0
Total failure	1	1
Specification	0	1
Loss of data	0	0
Security	0	0
Performance	0	0
Project management	0	0
Hotfix request	0	0
Implementation requests by component		
PS-Folio-AT	20	1
Web-FSCUIWIN:FSCUIWIN	2	2
Implementation requests by product planning folder		
PS-Folio-Projekte	21	1
Fabasoft Folio	4	6
Fabasoft Components Clients	7	7

6. RESULTS

The results of the evaluation of the two test executions for a software project were compared for this case study. Two comparisons were made.

The collected data in Table 1 has been graphically shown on pie charts and histograms in Microsoft Excel (see Figure 5, 6, 7 and 8) and analyzed with responsible project and product manager.

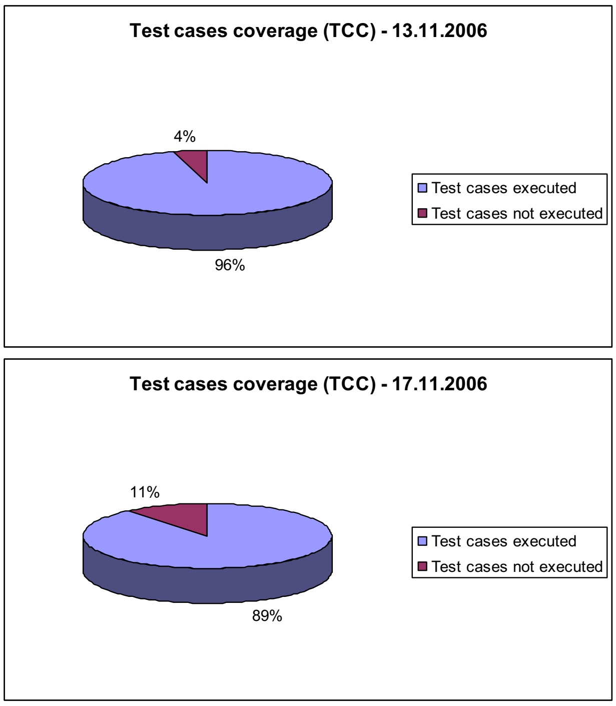


Figure 5. Test cases coverage

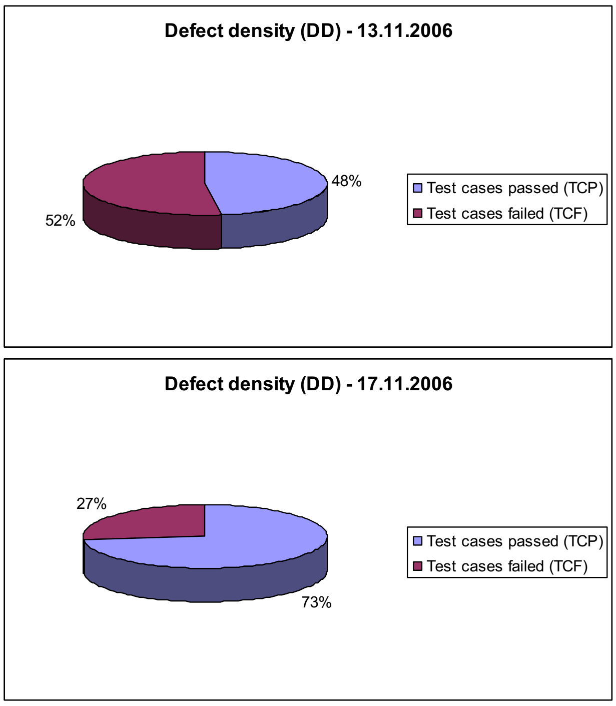


Figure 6. Defect density

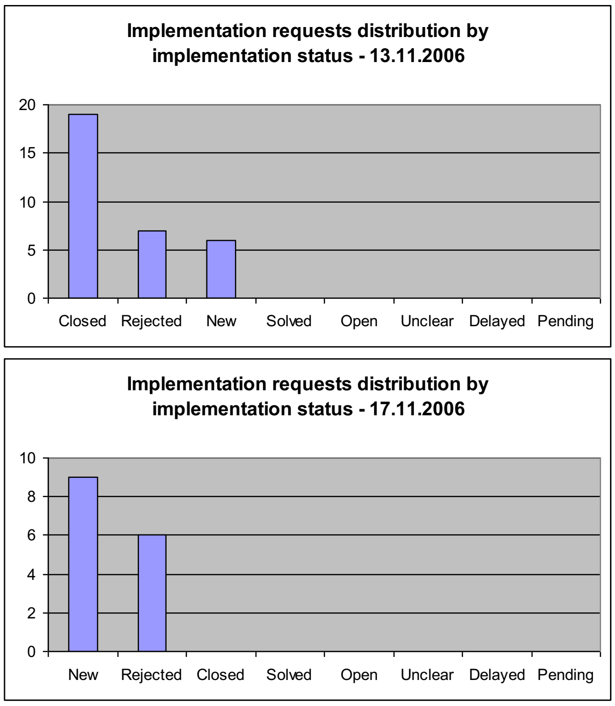


Figure 7. Implementation requests distribution by implementation status

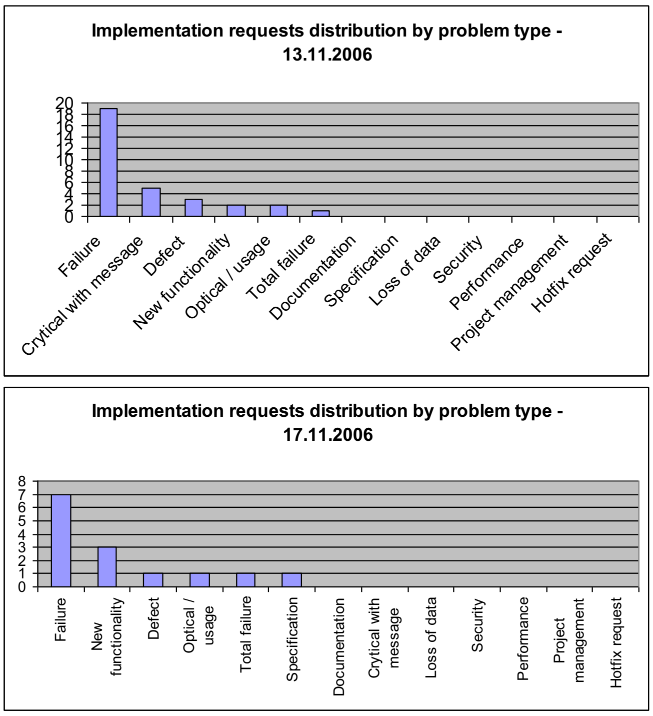


Figure 8. Implementation requests distribution by problem type

Pie charts which graphically represent test cases coverage are shown in Figure 5. In this example, TCC is not equal to 100% because some test cases were not executed and/or need to be clarified first. TCC provides the feedback to the test manager about the current state of testing progress.

Defect density is provided in Figure 6. In this example, DD shows that circa 52% test cases fails in the first test run and contains major or minor defects shown in Fig. 8. The second test execution shows significant defect density decrease.

Figure 7 shows the implementation requests distribution by implementation status. At the last test execution a lot of implementation requests have status „New" and need to be assessed by the responsible project/product manager. Many implementation requests with the implementation status „New" increase the number of defects released to production (DIP).

The test manager can use an example chart as shown in Figure 8. Presented histograms show the relation between the number of implementation requests and the problem type. The most detected defects are categorized as failure.

7. CONCLUSION

Generally, software projects still fail to be delivered on time, within budget, and with desired quality. A key point for reducing risks and quality improvement is to measure the quality of the product being developed, and become aware of potential problems.

Cost of defect correction is higher with time and most expensive if defects are detected by customers in production environments because end-users in their own right expect error free products. Test metrics can provide valuable information that is used in risk management, defect prevention and quality improvement during software development.

This paper used a simple test metrics model to show how test/project/product managers can use measures and metrics to better control software projects, identify risks, measure and increase software quality. This model is practical, may be applied to software projects, and help managers to have visibility into and control over their overall projects in addition to identifying and monitoring their risk areas.

At the beginning of software measurement implementation the best advice is to choose relevant, simple, and small set of test metrics that is appropriate for project, provide value to the organization and do not require a lot of effort to collect and analyze. Analyzing and interpreting the information produced by the test metrics is essential to making the right decisions. In the future test metrics model have to be extended with a new effective metrics to build an optimum test metrics model that provide useful information and helps managers for daily, weekly or monthly decision making.

8. ACKNOWLEDGEMENTS

This paper was produced as part of the Fabasoft quality assurance program. I am very grateful to Christian Kastner and Klaus Katterbauer who helped with their suggestions during the work on this paper.

Nadica Hrgarek is a member of the Research & Development eGov-Suite department at Fabasoft R&D Software GmbH & Co KG in Linz, Austria. She is a software quality engineer on Fabasoft eGov-Suite projects to ensure their success in using quality assurance methods, metrics and techniques. Her current research interests are statistical process control methods in software development process, software quality assurance, software metrics, and test management.

Hrgarek received a BS in information systems and a MS in information science from the University of Zagreb, Croatia. She is a member of the ASQF society.

REFERENCES

[1] Chidamber, S. R.; Kemerer, C. F. *A Metrics Suite for Object Oriented Design*, IEEE Transactions on Software Engineering, Vol. 20, No. 6, 1994, pp. 476-493.

[2] IEEE Computer Society. *IEEE Std. 1028:1997 IEEE Standard for Software Reviews*, New York, 1998.

[3] IEEE Computer Society. *IEEE Std. 610.12:1990 IEEE Standard Glossary of Software Engineering Terminology*, New York, 1990.

[4] IEEE Computer Society. *IEEE Std. 1061:1998 IEEE Standard for Software Quality Metrics Methodology*, New York, 1998.

[5] Kan, S. H. *Metrics and Models in Software Quality Engineering*, Second Edition, Addison-Wesley, Boston, 2004.

[6] Konda, K. R. *Measuring Defect Removal Accurately*, Software Test & Performance, Vol. 2, No. 6, July, 2005, pp. 35-39.

[7] Myers, G. J. *Methodisches Testen von Programmen*, Oldenbourg Wissenschaftsverlag GmbH, München, 2001.

Received: 11 January 2007

Accepted: 27 November 2007