Using Functional Defect Analysis as an Input for Software Process Improvement: Initial Results

Tanja Toroi[1], Anu Raninen[1,2], and Hannu Vainio[1]

[1]University of Eastern Finland, School of Computing, Kuopio, Finland
{tanja.toroi, anu.raninen, hannu.vainio}@uef.fi
[2]Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland

Abstract. In this paper we present how functional defect analysis can be applied for software process improvement (SPI) purposes. Software defect data is shown to be one of the most important available management information sources for SPI decisions. Our preliminary analysis with three software companies' defect data (11653 defects in total) showed that 65% of all the defects are functional defects. To better understand this mass, we have developed a detailed scheme for functional defect classification. Applying our scheme, defects can be classified with accuracy needed to generate practical results. The presented scheme is at initial stages of validation and has been tested with one software company's defect data consisting of 1740 functional defects. Based on the classification we were able to provide the case organization with practical improvement suggestions.

Keywords: functional defects, defect data analysis, process improvement

1 Introduction

Software defect analysis is recognized as an effective and important approach to software process improvement (SPI) [1]. Robert Grady has stated that defect analysis, tracking and removing the major sources of defects offer the greatest short-term potential for improvements [2]. However, despite its importance the defect data is rarely utilized in process improvement efforts of software companies [3].

Previous research has shown that the classification of defects is important when aiming at measurement-based process and product improvement [4]. In addition, the defect classifications can be used to identify product and process problems [5] and to improve the testing and/or inspection activities [6]. There are numerous defect classification schemes available in the literature. To name a few, IEEE provides a Standard Classification for Software Anomalies [7] and IBM has generated Orthogonal Defect Classification (ODC) [8]. In addition, Beizer [9] and Humphrey [10] have presented their defect classification schemes. Unfortunately, for our purposes, defect classification schemes published are too general in nature and classify defects at a rough level.

Our preliminary analysis with three software companies' defect data (11653 defects in total) showed that 65% of the defects stored in the companies' databases are

functional defects, i.e. defects in computation and/or functional logic [11]. In order to be able to use the defect data for process improvement purposes the functional defects had to be understood in more detail. To accomplish this, a more detailed defect classification of the functional defects was necessary to be conducted. However, there are not many defect classification schemes available for this purpose.

Beizer has defined a defect taxonomy [9] in which functional defects are divided in seven subclasses. We applied Beizer's functional defect classification for one software company's defect data consisting of 1740 functional defects. After applying Beizer's classification we noticed that over half of the functional defects (58%) were situated in one defect subtype, Feature/Function correctness. These results were not very useful in practice; over half of the defects remain in a single class. Hence, the functional defect classification was not detailed enough to identify the main problem areas.

In this paper, we present a detailed scheme for the classification of functional defects. The detailed classification scheme is an initial version based on analyzing defect data from one software company, including 1740 functional defects. Applying the classification scheme was encouraging: the result was easily recognizable inputs for process improvement. It appears that applying our scheme, the problems areas of software development and testing processes can be identified. Hence, testing can be focused on certain major issues. In addition, process improvement actions can be justifiably targeted to the problematic areas identified based on the defect data classification.

The aim of this paper is to present the initial results of applying our functional defect classification scheme and make the scheme available for other researchers and practitioners. We have already received feedback and improvement suggestions from our first case organization and are currently validating the scheme with the more data from other companies.

The overall structure of this paper is: Research setting is described in Section 2. In section 3, we present the general defect classification scheme, Beizer's functional defect classifications and our own scheme. Section 4 describes the results of applying the defect classifications. Section 5 gives process improvement suggestions based on functional defect data analysis. The results are discussed in section 6 and section 7 provides the conclusion.

2 Research Setting

It is shown that software defect data is one of the most important available management information sources for software process improvement decisions [2]. We conducted a preliminary study in spring in 2011 to find out what the most common defect types are and how this information can be used in process improvement [11]. The study was conducted using defect data from three software companies consisting of 11653 defects in total. Based on the results of the preliminary study it was noticed that further research was needed. The defect classification scheme applied was too

general in order to provide detailed information to be applied for process improvement purposes.

The initial study presented in this paper was conducted in one software company in the beginning of 2012. The case organization of the study is a Finnish software company with 18 employees. The organization has 9 employees in development and maintenance and 4-6 in testing. The company produces commercial off-the-shelf (COTS) products. An open source, web-based defect tracking system Mantis[1] is used in the company.

The results of the preliminary study conducted in 2011 showed that over half of the defects stored in the defect databases are functional defects (65%). In order to utilize defect data for process improvement purposes, functional defects had to be understood in more detail. Hence, the research problem of the study is: How functional defects should be classified so that the result provides practical inputs for software process improvement? In addition, we wanted to test our functional defect classification scheme and make the scheme available for other researchers and practitioners.

3 Functional Defect Classification

In this section we present the general defect distribution scheme applied in our previous study [11]. In addition, we present the functional defect classification by Beizer [9] and our own more precise initial scheme based on it.

3.1 General Defect Distribution Scheme

The defect distribution scheme applied in our preliminary study [11] is presented in Table 1. The scheme is a combination of the schemes by Beizer [9] and Humphrey [10]. It divides defects in ten types. We applied the scheme for three software companies defect data consisting of 11653 defects (see Section 4.1). The most common defects in every company were functional defects (65%), i.e. defects in computation and/or functional logic. In order to find out the real problems behind these functional defects they had to be investigated in more detail.

Table 1. General defect distribution scheme applied

ID	Defect Class	Description
1	Assignment	Declaration, duplicate names, scope, limits
2	Build, package, environment	Change management, library, version control
3	Checking	Error messages, inadequate checks
4	Data	Database structure and content
5	Documentation	Comments and messages
6	Function	Logic, pointers, loops, recursion, computation,

[1] http://www.mantisbt.org/

		function defects
7	Integration	Integration problems, component interface errors
8	Requirements	Misunderstood customer requirements
9	System	Configuration, timing, memory, hardware
10	User Interface	Procedure calls and references, I/O, user formats

3.2 Beizer's Taxonomy for the Functional Defects

In the literature, only a few functional defect taxonomies exist. In order to classify the functional defects in a more detailed manner, we applied Beizer's taxonomy [9] which has seven subcategories for functional defects. In addition to the functional defects, Beizer's taxonomy also includes structural defects. Structural defect type includes "Control Flow and Sequencing" (e.g. path left out, unreachable code, improper nesting loops) and "Processing" (algorithmic, arithmetic expressions, initialization) defects. We added the structural defect types to the classification because control flow and sequencing, and processing defects are actually quite similar to functional defects. Often failures that are caused by a sequencing defect appear as erroneous system functionality. Hence, the failure is entered into the defect database as a functional defect. Based on our experience, the defect types or descriptions of the defects are seldom altered after being entered to the database. The Beizer's taxonomy of the functional and structural defects is presented in Table 2.

Table 2. Taxonomy of the functional and structural defects [9]

ID	Defect type	Description
21xx	Feature/ Function correctness	Feature not understood, feature interaction
22xx	Feature Completeness	Missing feature, duplicated, overlapped feature
23xx	Functional Case Completeness	Missing case, duplicated, overlapped case, extraneous output data
24xx	Domain bugs	Domain misunderstood, boundary location error, boundary closure
25xx	User Messages and Diagnostics	False warning, failure to warn, wrong message, spelling, formats
26xx	Exception Condition Mishandled	Exception conditions are not correctly handled, wrong exception-handling mechanisms used
29xx	Other functional bugs	Other functional bugs that are not mentioned in the previous rows.
31xx	Control Flow and Sequencing	(Structural bug) Path left out, unreachable code, improper nesting loops, loop termination criteria incorrect
32xx	Processing	(Structural bug) Algorithmic, arithmetic expressions, initialization, cleanup, precision

3.3 Improved Functional Defect Classification Scheme

The main problem with applying Beizer's taxonomy was that it is not detailed enough to identify the practical targets for process improvement. The defect type "Feature/Function correctness" included most of the defects in the end. In addition, a "Feature completeness" defect is often hard to distinguish from a "Function/Feature correctness" defect. Further, due to the nature of the defect data analyzed, a "Functional case completeness" defect was quite impossible to detect.

To avoid the problems stated above, we developed a more detailed scheme in which a "Feature/Function correctness" defect type is divided into subtypes. In addition, we added "Control flow and sequencing" and "Processing" defect types to our functional defect scheme. Further, in our scheme "Domain bugs" refer to application domain defects not value ranges of the variables. Our initial functional defect classification scheme is presented in Table 3.

Table 3. Initial functional defect scheme

6	**Functional defect type**	**Description**
6.1	Control flow and sequencing	Defects in control flow (e.g. path left out, unreachable code, improper nesting loops, loop termination criteria incorrect)
6.2	Domain Bugs	Application domain bugs, subcategories vary between companies (e.g. taxes, allowances, materials)
6.3	Exception condition mishandled	Defects in exception handling.
6.4	Feature Completeness	Feature is executed inadequately.
6.5	Feature / Function Correctness	Implementation of feature / function is incorrect.
6.5.1	Copying data	Defects in copying data between systems / databases. Difficulties in making backups.
6.5.2	Default values and initial states	Defects in programs default values e.g. programs default selection causes failures in software.
6.5.3	Installation	Problems during installation of the developed program.
6.5.4	Retrieval, update and removal of data	Relates to refreshing the screen. Data inputs from user doesn't update properly to the screen.
6.5.5	Saving data	Data doesn't save to system. Data can't be saved when it should be possible or it can be saved when it shouldn't be able.
6.5.6	Utilizing operating system services	Problems related to operating systems (e.g. Windows), e.g. mouse commands, tab order, and other features provided by the OS.

6.6	Processing	Defects in processing, calculations.
6.7	User messages and diagnostics	User messages are incorrect. Printing on screen / paper, defects in reports.

4 Applying Functional Defect Analysis in Process Improvement

In this Section we present the results of the defect classification after the first general classification, after applying Beizer's taxonomy, and after applying our initial functional defect scheme. In addition, improvement suggestions collected from the case organization are discussed.

4.1 General Defect Distribution

In our preliminary study [11] we applied the general defect classification scheme presented in Table 1 for three software companies defect data consisting of 11653 defects. The result of the defect classification is presented in Figure 1. From the Figure, it can be seen that by far the most common defect type in every company is "Function" defect type (total of 7574, 65%). The second most common defect types are "User Interface" (total of 1870 defects, 16%), "Assignment" (total of 700 defects, 6%) and "Checking" (total of 688 defects, 5.9%). "Requirements" (total of 24 defects, 0.2%) and "Documentation" (total of 47 defects, 0.4%) are the rarest defect types.

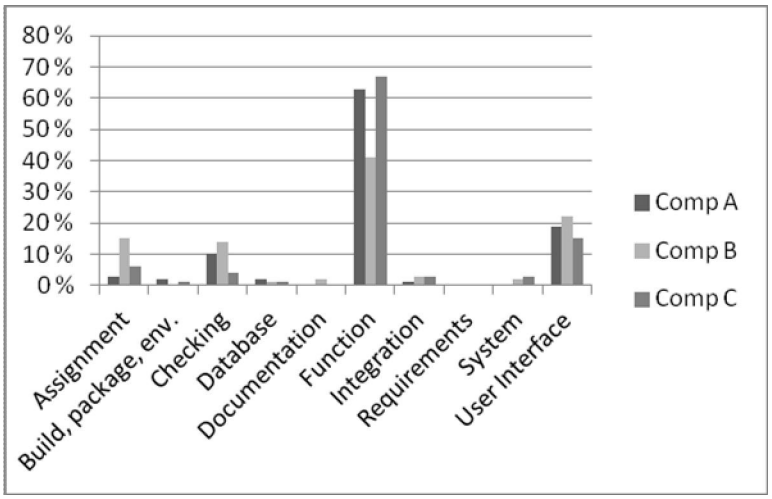

Fig. 1. Defect distribution after the first classification

4.2 Functional Defects Classified According to Beizer's Taxonomy

In order to make the defect classification data usable in practice, we needed to better understand what the mass of functional defects consisted of. Hence, we applied func-

tional defect taxonomy by Beizer [9] to classify the defects in a more precise manner. The preliminary classification was conducted for one software company's defect data consisting of 1740 functional defects.

The defect distribution is presented in Figure 2. In practice, we ended up formatting Beizer's taxonomy. We did not include "Other functional defects" because this is too vague to tell anything about the defects nature. In addition, we did not identify a single "Functional case completeness" defect. This may be due to the cursory description of the defect type. From the Figure 2, it can be seen that the defect type "Feature/Function correctness" is remarkably more common than the other defect types. "Feature/Function correctness" includes 58% of the defects. The rest of the defect types include evenly from 4 to 13 percent of the defects. One exception is "Exception condition mishandled" type which includes only 0.29% of the defects.

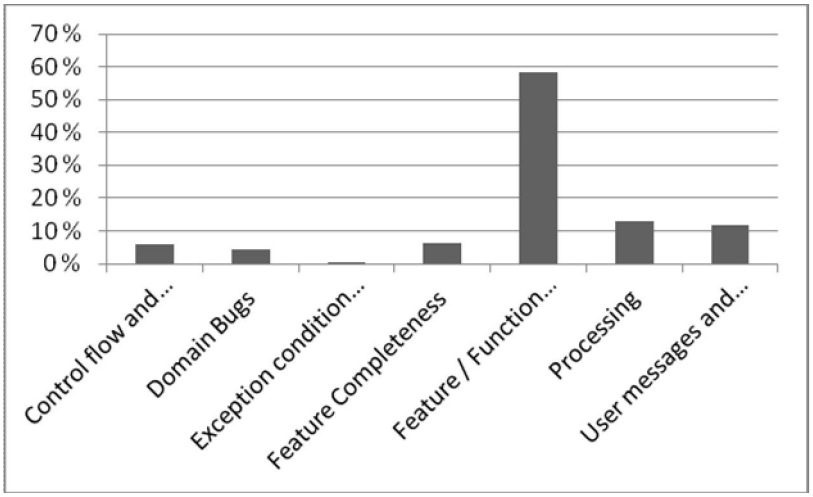

Fig. 2. Functional defect distribution classified according to Beizer's taxonomy

4.3 Functional Defects Classified According to Our Own Defect Scheme

The defect classification according to Beizer's taxonomy was still not detailed enough for process improvement purposes of the case organization. They wanted to find out what the "Feature/Function correctness" issues are, in order to improve their development and testing processes. In order to figure this out, we defined a more detailed scheme for the functional defects. The scheme is presented in Table 3. We applied the scheme for the same 1740 defects as with the Beizer's taxonomy. The results can be seen in Figure 3.

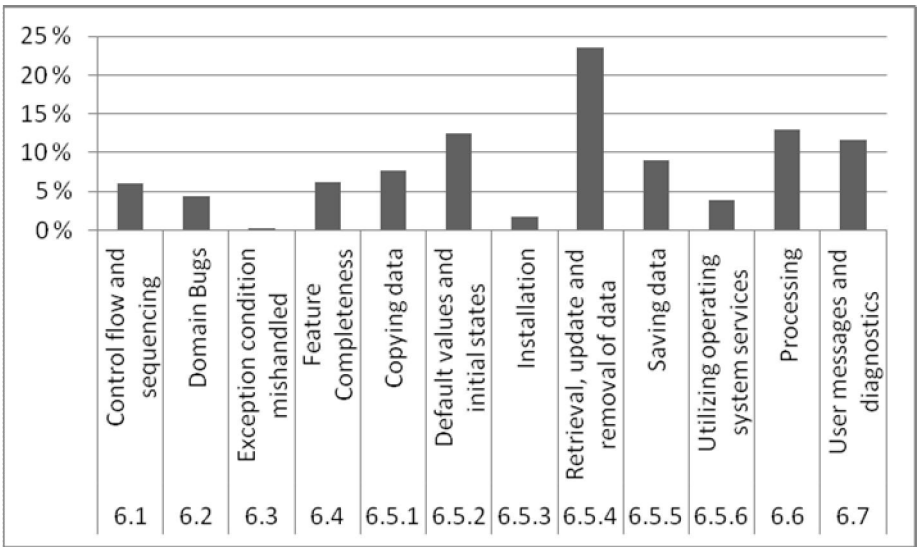

Fig. 3. Functional defect distribution classified according to our own scheme

The distribution of the defects is notably more even applying our scheme. The most common functional defect type is "Retrieval, update and removal of data" (24% of the defects). The second most common defect types are "Processing" (13%) and "Default values and initial states" (13%). "Exception condition mishandled" is the most uncommon defect type (only 0.3% of the defects).

4.4 Further Development of the Functional Defect Scheme

The functional defect scheme presented in this paper is an initial version and applied only to one software company's defect data. The goal of the classification scheme is to be general enough to be applied in most software companies. To achieve this, the classification scheme must be documented unambiguously. In addition, the defect types must be general enough to be applied in several companies. Yet, the defect types must be particular enough to get significant results out of the classification exercise. In order to improve the functional defect classification scheme, we collected feedback from our case organization.

The feedback was collected in a three hour workshop organized in the premises of the case organization. In the workshop the classification scheme and result of the classification were presented for the participants from the case organization and then discussed in detail.

Mainly the case organization was happy with the classification scheme and the result of the classification. However, there were two defect types that caused confusion. The case organization had problems understanding the defect types "6.5.6 Utilizing operating system services" and "6.5.2 Default values and initial states". The representatives of the case organization questioned whether these two types were already included in the higher level of the classification. They wanted to know what was the

difference between “6.5.6 Utilizing operating system services” and “10 User Interface” defect types. Also, they wanted to know why “6.5.2 Default values and initial states” defects were not included in the “4 Data” type.

Based on the feedback the description of these two defect types was written in more detail. The improved descriptions that highlight the difference between these defect types are presented in Table 4.

Table 4. The difference between defect types Data & Default values and initial states and User Interface & Utilizing operating system services

Defect Type	**Description**
4 Data (see Table 1) vs.	Database structure and content. For example; bug due to error in the structure of the database, bug due to the availability of the data, bug due to difficulties in obtaining the data from the database.
6.5.2 Default values and initial states (see Table 3)	Defects related to default values and initial statuses of the software. Default values or initial states that prevent the user from using the system as intended. For example; the user is presented with wrong and/or wrong sized screens as a default.
	6.5.2 Distinguishes from type 4: Default values and initial states are different from data defects as they are regarded different by nature. All default values do not necessarily derive from database.
10 User Interface (see Table 1) vs.	Procedure calls and references, I/O, user formats. For example; incorrect output data from the user point of view, a problem with usability and/or trivial defect in layout (e.g. overlapping windows)
6.5.6 Utilizing operating system services (see Table 3)	Defects related to utilizing the services of the operating system of the computer on which the software is installed. For example; defects due to applying the monitors, printers and other peripherals. A defect related to Windows system (e.g. tab-order).
	6.5.6 Distinguishes from type 10: User Interface defects are more often cosmetic defects, for example, typing errors in user interface.

5 Process Improvement Suggestions Based on Functional Defect Data Analysis

Based on the functional defect data classification, the case organization is able to see their software engineering problem points from the defect point of view. The classifi-

cation shows that the most troublesome issues are related to retrieving, updating and removing data, default values of the variables and forms, processing i.e. calculation, and user messages and diagnostics. The most common functional defect types of the case organization are presented in Figure 4.

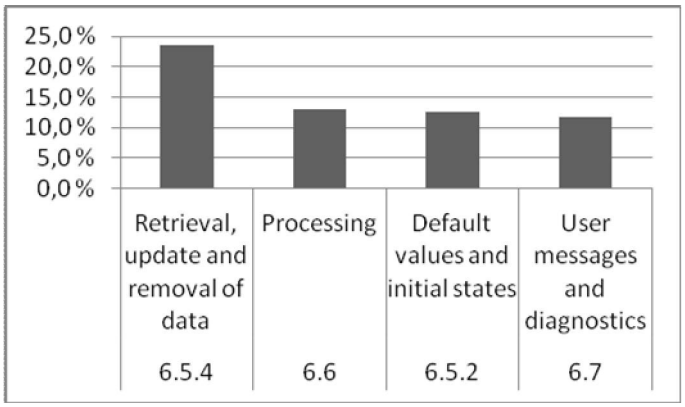

Fig. 4. The most common functional defect types in the case organization's defect data

Based on the results of the defect classification improvement suggestions were given to the case organization. The suggestions are presented in Table 5.

Table 5. Improvement suggestions related to the most common defect types

Defect Type	**Improvement suggestions**
Retrieval, update and removal of data	**Stress the importance of unit testing**. Data retrieval, updating and deletion defects could be detected already in the unit testing phase during which it would be cheaper to fix them. **Conduct pair programming**. Previous research has found that programmers working in pairs produce fewer bugs, than programmers working alone [12].
Processing	Conduct **code inspections** in order to reduce the amount of bugs due to carelessness. Processing bugs are often due to the software engineer not being careful enough while coding the calculation rules to the software. Inspection is proved to be effective at identifying defects [13].
Default values and initial states	Conduct **code inspections and pair programming**. **Take test automation in use.** Test automation does not prevent the defects but would make it easier and more cost-effective to detect them from the code [14].
User messages and diagnostics	**Conduct usability testing.** This could help to find defects in user messages. **Inspect** end user reports in order to find anomalies and bugs in them.

6 Discussion

Our preliminary analysis with three software companies' defect data (11653 defects) showed that 65% of the defects were functional defects [11]. We wanted to find out what the real problems are behind these functional defects in order to enable process improvement based on defect data. Defect data is one of the most important available management information sources for software process improvement decisions [2]. Yet, defect data is rarely utilized properly in process improvement efforts [3].

However, in the literature, there are only a few functional defect classifications available. Beizer has developed a defect taxonomy which has subcategories for functional defects [9]. We applied Beizer's taxonomy for functional defect data (1740 functional defects). However, the results were not satisfying, over half of the defects still remained of one defect type, "Feature/Function correctness". Beizer's taxonomy was not able to properly make the problem areas of the process visible.

The main problem with applying Beizer's taxonomy was that it is not detailed enough to identify the tangible targets for process improvement. Namely, the "Feature/Function correctness" defect type is so general that far too many of the defects are of this type. In addition, "Feature completeness" type is often impossible to distinguish from the defect type "Function/Feature correctness". When the defect has been entered to the database it cannot often be known whether the feature causing a defect has been properly completed or incorrectly coded. Further, a "Functional case completeness" defect is quite difficult to identify from the defect data.

To avoid the problems stated above and to better identify the problem areas of the processes we defined a more detailed functional defect classification in which the defect type "Feature/Function correctness" is refined in more detail. We applied our scheme for one software company's functional defect data and received a more diversified defect distribution. Based on the functional defect analysis, practical process improvement suggestions could be provided. It was suggested that the company should conduct code inspections to identify simple errors earlier. In addition, they should stress the importance on unit testing to the programmers.

Further, the results of the functional defect data classification can be utilized in making decisions on whether testing should be automated. It is important for the test team to manage automated testing expectations and to outline the potential benefits of automated testing [14]. Overall, the functional defect analysis can be used in justification when more resources for verification and validation processes are required.

7 Conclusion

In this paper we have presented how functional defect classification can be applied as an input for process improvement. A functional defect classification scheme is presented and applied for one software company's defect data (1740 functional defects). Based on the results of the defect analysis, process improvement suggestions are provided. Applying our scheme, the problems areas of the development and testing processes can be identified and testing can be focused on certain major issues. In addi-

tion, process improvement actions can be targeted to the areas identified based on the defect data classification.

Our scheme is an initial version. Due to promising results reached applying it to one company's data we are currently validating it via applying it to additional companies' defect databases and collecting feedback from the companies.

Acknowledgements

This research was funded by the Finnish Funding Agency for Technology and Innovation (Tekes) with grant 70030/10 for METRI (Metrics Based Failure Prevention in Software Engineering) project and supported, in part, by Science Foundation Ireland grant 03/CE2/I303 1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

1. Vinter, O.: Experience-Based Approaches to Process Improvement. In: Proceedings of the 13th International Software Quality Week, San Francisco, USA (2000)
2. Grady, R. B.: Practical software metrics for project management and process improvement. Prentice Hall, New Jersey (1992)
3. Fredericks, M., Basili, V.: Using Defect Tracking and Analysis to Improve Software Quality. DoD Data & Analysis Center for Software (DACS) (1998)
4. El Emam, K., Wieczorek, I.: The repeatability of code defect classifications. In: Proceedings of the Ninth International Symposium on Software Reliability Engineering, pp. 322-333 (1998)
5. Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K.., Atkinson, J. S., Lepori-Costello, C., Jasper, P. Y., Tarver, E. D., Lewis, C. C., Yonezawa, M.: In-process improvement through defect data interpretation. IBM Systems Journal, 33, 1, 182 –214 (1994)
6. Freimut. B.: Developing and using defect classification schemes. Fraunhofer IESE IESE-Report No, 72 (2001)
7. Ieee standard classification for software anomalies. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), pp. C1-15, 7 (2010)
8. Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., Wong, M.-Y.: Orthogonal defect classification – a concept for in-process measurements. Software Engineering, IEEE Transactions on, 18, 11, 943-956 (1992)
9. Beizer, B.: Software Testing Techniques. International Thomson Computer Press (1990)
10. Humphrey, W.: A discipline for software engineering. Addison-Wesley (2007)
11. Raninen, A., Toroi, T., Vainio, H., Ahonen, J.J,: Defect Data Analysis as Input for Software Process Improvement, TBP 13th International Conference on Product-Focused Software Development and Process Improvement, PROFES 2012 (2012)
12. Cockburn, A., Williams. L.: The Costs and Benefits of Pair Programming, In: Succi, G., Marchesi, M. (eds.) Extreme programming examined, pp. 223-243 (2001)
13. Gilb, T., Graham, D.: Software inspection. Addison-Wesley, Great Britain (1993)
14. Dustin, E., Rashka, J., Paul, J.: Automated Software Testing: Introduction, Management, and Performance. Addison-Wesley (1999)