Bug Fixing Process Analysis using Program Slicing Techniques

Raula Gaikovina Kula and Hajimu Iida

Graduate School of Information Science, Nara Institute of Science and Technology

Takayamacho 8916-5, Ikoma, Nara 630-0101, JAPAN

raula-k@is.naist.jp, iida@itc.naist.jp

Abstract

This paper explores alternative assessments of Software Process by analysis of fine grain processes of bug fixing with their bug characteristics. Using code metrics to the program slices for bugs, the research studied the relationships between bug characteristics and their bug fixing processes. Lines of code(LOC) and Code complexity (CC) were used for initial evaluation.

1. Introduction

1.1 Background and Related Work

The quality and improvement of software processes are seen as vital to any software development project. Traditionally software processes usually refers to the main phases in the software development life cycle. Improvement of these processes is a major activity for larger software organizations as benefits are seen in the cost and business value of improvement efforts, as well as the yearly improvement in productivity of development, early defect detection and maintenance, and faster time to market [1].

There have been a number of studies on the issues relating to the implementation of software process assessment / assurance models such as CMMI[2] and ISO 9000 [3] [4] [5] [6]. Much of the issues lie with the size of the organization as these models are expensive to assess and implement from small software organization [7]. These studies show that the processes assessments are sometimes tedious and usually not tailored for specific companies. Also additional studies show that higher management support, training, awareness, allocation of resources, staff involvement, experienced staff and defined software improvement process implementation methodology is critical for software process improvement [8]. There has been several related work into trying to make the models easier and better to use. Yoo [9] suggests a model that combines CMMI and ISO methods. Amburst [10] takes a different approach by treating software as manufacturing product lines, therefore making the processes systematic and generic.

An alternative measure of software processes can be done through the inspection of process execution histories. This analysis however is performed at the developer's level, and measures the fine grained processes called 'micro processes'. One such research has been done by Morisaki el al [11]. This research analyzes the quality of micro processes by studying the process execution histories such as change logs. Quality of the process is measured by the execution sequence, occurrences of missing or invalid sequences.

1.2 Research Motivation and Objective

Our motivation is to present a method of quantitative and handy software process assessment, focused to help efficiency of the workload of developers. Current models of process assessment are at the software life cycle level and require complicated assessment and are also expensive as they require highly trained assessors. The author's ambition is to work towards models that improve and assess quality of micro processes. More specifically, a proof of concept towards a prediction model based on estimating the bug processes execution time will be the contribution of this research towards improvement of software processes at a micro level.

It is envisioned this study may aid developers in the micro process of bug fixing. The concept is that by aiding the developer with tools to better manage bugs, it may lead to better quality in software processes at this level, as well as influencing the overall software development processes.

The main objective of paper is to provide a proof of concept that there is indeed a relationship between a bugs attributes and how its related processes are executed. For example, a bug that has a lengthy process has certain characteristics as compared to a bug that is easily fixed in just a couple of days. Based on this objective, the following hypotheses are presented for testing:

- *Hypothesis 1:* Similar bug programs fragments shares similar characteristics of micro process execution
- *Hypothesis 2:* Deviant micro process execution occur due to some sort of bug classification

Section 2 presents the methodology, which includes the terms and definitions used in the research as well as the proposed approach. Section 3 then explains the experiment using the approach and tools to carry out the experiments. Section 3 also presents the results of the experiments. Section 4 is a discussion and analysis of the results as well as the validation and application of the research. Finally, section 5 outlines the future work followed by the conclusion.

2.Methodology of Debug Process Analysis

As mentioned in the research objectives, the research aims to find a relationship between a bug characteristic and the processes used to fix the bug. We attempt to first classify bugs according to the characteristics of the code. Program slicing techniques have been applied to identify certain characteristics of bugs.

2.1 Bug Fixing Process Definitions and Analysis

This research focuses on the day to day processes performed in the development of software. This paper assumes data repositories including following two locations in which bug information is stored daily.

1)Bug Tracking System

Typically a software development team uses a system to manage bugs in a medium to large scale project. The tracking system usually tracks progress of bug. This research utilizes the bug tracking systems to gather various properties of the bug.

2)Source Code Repository

The source code repository refers the system manages changes to source code in a software project. The two main system used is the Subversion (SVN) and Concurrent Versioning System (CVS). For this research, both SVN and CVS repositories were used to gather data on bug characteristics.

We also employ the generalized model of micro processes of bug tracking system proposed by Morisaki. The research shows that bug fixing comprises of these steps illustrated in Figure 1.

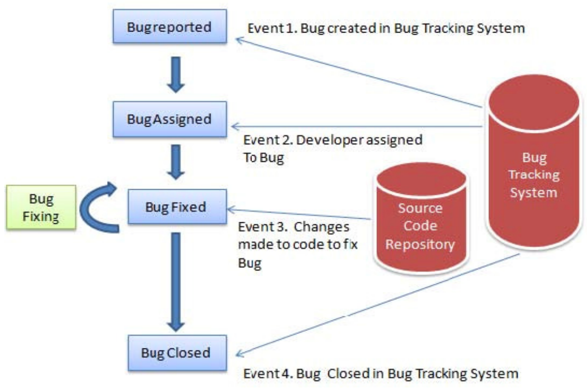

Bugreported
Event 1. Bug created in Bug Tracking System
Bug Assigned
Event 2. Developer assigned To Bug
Bug Fixing
Bug Fixed
Event 3. Changes made to code to fix Bug
Source Code Repository
Bug Tracking System
Bug Closed
Event 4. Bug Closed in Bug Tracking System

Figure 1. Bug Fixing Process Model

The quality of the micro process analysis is evaluated by how well the different phases are executed. Deviations such as incorrect sequences and omissions are used to indicate the low quality of micro processes of a bug fixing in a software project.

2.2 Characterization of Bug Code with Program Slicing Metrics

To assess the bug characteristics, program slicing metrics is used in this approach to describe bug characteristic metrics. Program slicing is a concept first described by Weiser [12] as a method to study the behavior of source code through the flow dependency and control dependency relationship among statements. This work however has been used in the field of Software Evolution [13]. Previous work has been done on using program slicing metrics to classify bugs. Pan et. al. [14] showed proved that program slicing metrics work as well as conventional bug classification methodologies.

Since it is the initial research on this approach, two of the basic program metrics, Lines of Code (LoC) and Cyclomatic Complexity (CC) [15] are employed to express characteristics of the program slices. The analysis is done at the function level with both metrics.

2.3 Analysis Steps

The analysis of the bug process includes following three steps:

1) Micro process extraction and analysis:

This step involves data mining and extraction of bug related attributes from both the source code repository and the bug tracking system. Table 1 and 2 shows the data that is extracted from the bug tracking system and the data extracted from source code repository, respectively.

2) The program slice extraction and analysis:

This step involves analysis of the code using the program slicing metrics. The following explains in detail how each metric was used to for bug classification.

- Bug LoC (Lines of Code) : This metric is proposed to measure how much of the code is potentially affected by the bug. This will be measured using the program slicing metric Lines of Code (LoC)' as

defined in section 2.2.1. The range of a bug is summarized in the equation below:

$$BugCC = \sum_{f \in \text{affectedfiles}} LoC(f)$$

where *affectedfiles* refers to the source code files that were affected when the bug was being fixed, and *LoC(f)* is lines of code of file *f*. The affected files include files that are dependent code related to the files edited during the bug fix.

- Bug CC (Cyclomatic Complexity) : This metric is proposed to calculate the potential complexity of the code affected by the bug. This will be measured using the program slicing metric CC as explained in 2.4.2. The severity of a bug is:

$$BugCC = \sum_{f \in \text{affectedfiles}} CC(f)$$

where *affectedfiles* refers to the source code files that were affected when the bug was being fixed, and *CC(f)* is cyclomatic complexity of file *f*. Since CC is calculated per function, Bug CC is sum of all the cc for each function in the affected files. The affected files include files that are dependent code related to the files edited during the bug fix

3) Total Analysis of Data

Using the MPA data extracted and the program slicing metrics extracted from the software project, the bugs will be grouped according to similar program slicing metrics characteristics. For this research, since the research is focused on the quality of micro processes involved in bug fixing, the analysis involves identifying how program slicing affects the execution of micro processes of a bug.

Table 1 Data extracted from the bug tracking system

Attribute	Description
Revision ID	Identification number used to track changes made to source code.
Bug ID	This is the reference to what changes where related to a bug.
Commit Date	This is the date when the code change was performed.

Table2 Data extracted from source code repository

Attribute	Description
Bug ID	This is used to identify the bug
Date Opened	This is when the bug was reported
Date Closed	This is when the bug was reported as fixed.
Bug Priority	This could be used to help understand its perceived importance.

Table3. Bug related data extracted by the tool

Data Repository	Bug Attribute	Description
Source Code Repository	Revision ID	Revision Identification number
	Date of Bug Fix	When the files related to the bug fix was edited.
	Bug ID	Bug Identification number
Bug Tracking System	Bug ID	Bug Identification number
	Create Date	Date when bug was created
	Assigned Date	Date when developer was assigned to bug
	Closed Date	Date when the bug was fixed
	Priority	Priority level of the bug

3. Experiment

We have conducted an initial experiment using open source software repositories. The main reason why we choose OSS is that it is easily accessible as well as does not require special permission or software licenses to analyze datasets. Also there is a large range of software projects available.

3.1 Tools

We used following tools selected for both the program slicing and the tool used to extract the MPA data:

Program Slicing Tool

Based on previous surveys [16] on the available program slicing tools, three program slicing tools[17][18][19] were tested and the most appropriate tool selected. In the end CodeSurfer[19] was chosen because it met the needs of the research as well as being used in Pan's use of program slicing metrics for bug classification. The main limitation of

CodeSurfer is that the projects to be analyzed can only be developed in the C, C++ programming language.

Bug Extraction Tool (Extract the MPA Data)

Using the micro process model we designed a bug extraction tool that would help search and extract the needed data related to this research. Published tools such as Kenyon [20] are available, however because data to be extracted are specific to this study, it was justified to develop the tool in-house.

Our extraction tool which was developed in java, acted as a web spider searching the online data repositories, and then parsing extracted data. In order to make the tool more flexible, it was developed to search both SVN and CVS repositories. This would not become a constraint on the research.

Table 3 presents the bug related data extracted by the tool.

3.2 Test Subjects

Due to the limitations of CodeSurfer only able to analyze projects based on the C/C++ programming language, projects were selected based on program slicing capability and on quality of bug extraction. Using Sourceforge.net as a source of the Open Source Software projects, the following three software projects met the selection criteria:

- Scintilla: Scintilla is an editing component for Win32 and GTK+. It is used when editing and debugging source code.
- wxWidgets: WxWidgets is a C++ library that lets developers create applications for Windows, OS X, Linux and UNIX on 32-bit and 64-bit architectures. Like Filezilla, wxWidgets used subversion and houses its bug tracking system outside Sourceforge.net.
- Filezilla: Filezilla is the open source File Transfer Protocol solution. The client component of the project was used for analysis.

All project source code are hosted independently. Scintilla uses the Sourceforge Bug Tracker as its bug tracking system. WxWidgets and Filezilla both use the Trac issue tracking system5 to help manage their bugs.

3.3 Findings

For the classification of the bugs, a proposed analysis method was using how long it takes to fix a bug. The number of days taken to fix a bug is calculated as:

Days taken to fix a bug = Date of Bug Fix – Date of Bug Detected

where bug fix is the date when the code changes of the bug are committed to the code and bug detected being the date when the bug was first reported to the bug. Bug Fix was used instead of 'Bug Close' status as it is more accurate of when the bug was fixed.

The Figure 3 shows the distribution of all bugs. Since the distribution covers a very large range of data, logarithmic graphs were used to plot the data. It was noticed that the bugs from 0 to 10 days were evenly distributed. From 10 days onwards, the distance between the bugs seems to increase. So we identify these bugs are the one group from 0-10 days. It can be seen that around bugs that are older than 64 days have a wider distribution compared to the bugs fixed in less than 64 days. With this observation, the second group of bugs was proposed from 11 to 64 days and then the third group from 64 days onwards.

Micro Process

Process execution was divided into follwing three groups to assist with the analysis:

– *Fixed and Closed Same Day* : These groups of bugs

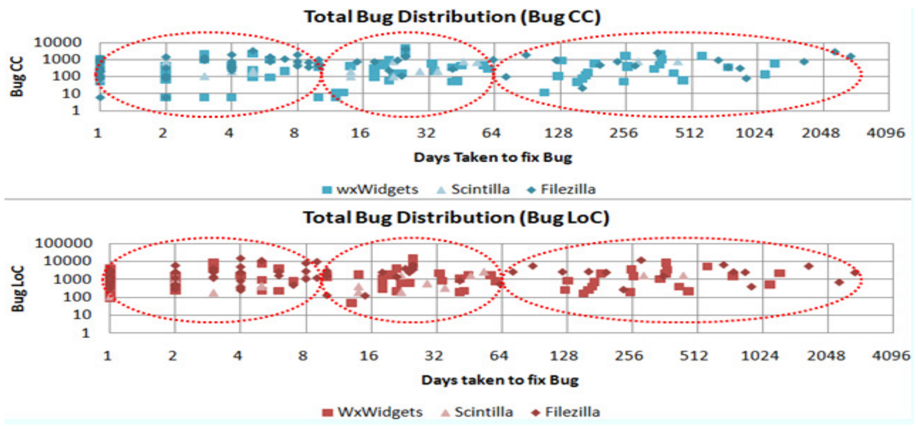


Figure 3. Total Bug Dstribution on log-log graph

contain bugs that are closed on the same day. It can be used to view bugs that were fixed but not closed just yet.

- *Standard Sequence* : This group contains the bugs that have all the processes sequenced in correct order.
- *Incorrect Sequence*: This group of bugs have incorrect sequences of the bug process compared the proposed model. The term 'incorrect' refers to abnormal sequence and behavior with the bug fixing process.

Bug Fixing Process using the Bug Classification

Results are summarized in Figure 4.

- Scintilla : The incorrect sequences occur in Bugs fixed in 0-10 days only. Also bugs that were fixed and closed the same day occur in all three groups. Finally bugs that took more than 64 days were all closed on the same day as being fixed.
- WxWidgets : There is an even distribution of the different types of bug fixing process executions, however, bugs fixed and closed on the same day always is greater than the other groups.
- Filezilla : The Figure indicates that the incorrect sequence, although very small occurs in bugs that took up to 10 days to fix and more than 64 days to fix. As with Filezilla, the bugs that were fixed and closed in the same day occur in all bug groups, however the most is found in bugs that took up to 10 days to fix. It is interesting to note that bugs that took 11 to 64 days did not have any incorrect sequence in its bug fixing process.

Program Slicing Metrics

- Bug CC Analysis: The graphs in Figure 5 illustrate the distribution of cyclomatic complexity of the bugs. The results indicate that the 11 to 64 days group has the largest distribution compared to the other groups. Also 11 to 64 days has higher bug CC compared as well. This suggests that bugs that are fixed in 11 to 64 days have a larger range of bug CC and are higher in complexity as compared to the other groupings. It is also interesting to note that bug that have more than 64 days have low complexity and a lower distribution.
- Bug LoC Analysis: The box plots illustrated in Figure 6 show the Bug LoC for bugs within the bug classifications. The graphs clearly show that bugs that take 11-64 days to fix have the highest lines of code in all projects. Apart from the highest lines of code we cannot make other similar characteristics. It seems that there is no clear trend from the three projects. Scintilla and wxWidgets seem to indicate that the 11 to 64 days has a largest distribution of Bug CC, however Filezilla suggest that maybe all groups have a similar distribution. These results indicate that further analysis with a larger test group to be able to make concrete analysis.

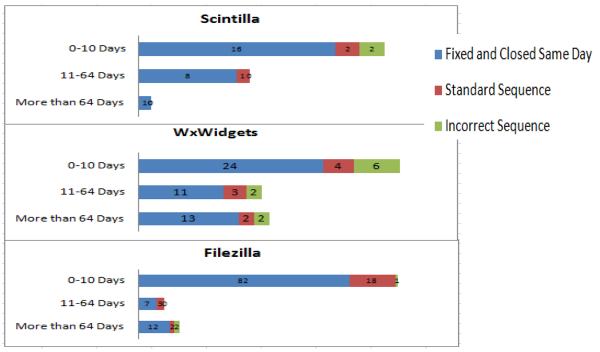


Figure 4. Bug fixing process compared with bug classifications

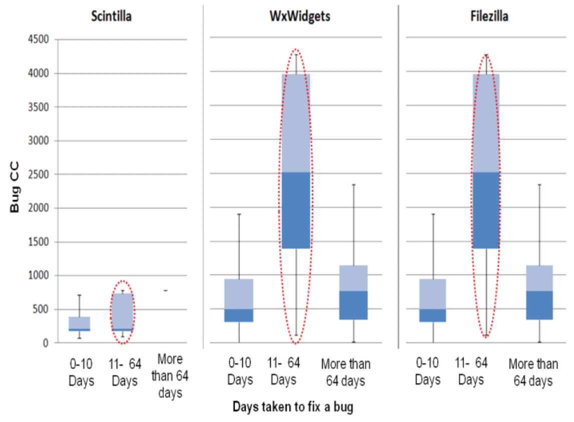


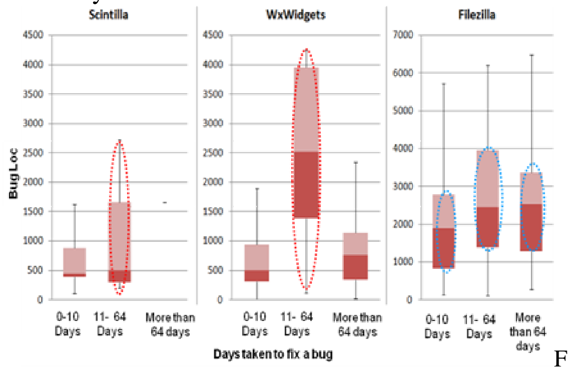
Figure 5. Bug CC Distribution using Bug Classification across Projects

Total Data Analysis

Finally, we performed an analysis combining both the micro process execution and program slicing metrics. Results are as follows:

- Bug CC: The graphs in Figure 7 are the distribution of Bug CC metric of bugs grouped according to the bug fixing process per project. The graphs suggest that in each project, bugs with incorrect sequences have the highest distribution and maximum bug CC. The interpretation could be misleading as seen in the previous section, incorrect sequences account for only 2% to 15% of the sample size. However shows that further analysis is needed and the direction is promising.

– Bug LoC: Figure 8 show the results of when the groupings of bugs according to execution process were measured using the Bug LoC metric. Similar to the results seen in Bug CC, Bug LoC also displays incorrect sequences as having the widest range and highest lines of code as compared to the other groups. Similar to Bug CC analysis with program slicing, this information could be misleading as the sample sizes for the incorrect sequences are extremely low.



F igure 6. Bug LoC Distribution using Bug Classification across Projects

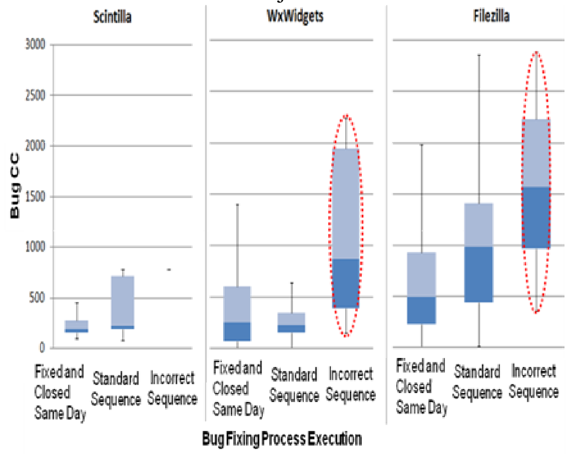


Figure 7.BugCC & Process Classifications

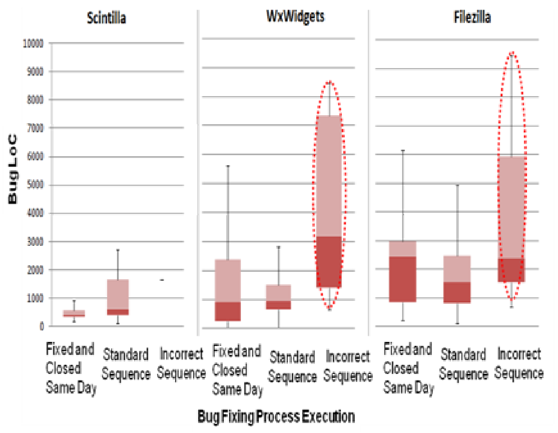


Figure 8.Bug LoC & Process Classification

4. Discussion

4.1 Findings in Analysis

The approach taken proves that bugs could be grouped based on program slicing based metrics. Using our proposed approach results firstly showed that using the program slicing metrics, the bugs were grouped according to how long it takes to fix the bug. With visual analysis of the distribution, it was found that the bug distribution changed after 10 days and 64 days.

Further analysis was done on the groupings resulting with many interesting findings. Firstly most bugs are fixed within ten days of being detected. Also priority is not a factor with the bug classification, as shown in this experiment Filezilla fixed very urgent bugs within 10 days while wxWidgets addressed urgent bugs after 64 days.

To better understand the bug classification, the distributions of program slicing metrics were analyzed according to the bug groupings. Findings suggest that bugs fixed within 10 days show lower complexity. Bugs between 11 and 64 days show a wider range of complexity and bugs that took more than 64 days had a much lower complexity. Since most bugs lie in the 0-10 day group, it can be concluded that bugs with lower complexity are fixed in less days. In relation of lines of code per bug, there seem some trends but nothing concrete enough to propose, therefore further analysis and review of the lines of code metric need to be performed.

The second part of the research aimed at analyzing the micro process execution, and grouped using the bug classifications. Results firstly show not all projects follow the proposed model for bug fixing. Both Filezilla and wxWidgets omitted the ‘assign to developer’ activity in the bug fixing process. Possible reasons were be the limitations of the bug tracking tool or the assignment process is not part of that projects bug fixing process related to two of the three projects. Also this shows the real time-frame in which the bug was fixed.

Another interesting finding was the appearance of bugs that were closed on the same day that it was fixed. In all test subjects, results indicated that the majority of bugs had bug closed the same day they were fixed. It was found that bugs closed in 0-10 days were most prone to errors in the bug fixing process. All projects indicated more bugs with incorrect sequences occur in bugs fixed in ten days. Bugs older than 10 days have a lesser chance of having incorrect sequences.

Finally, the main part of the analysis was done combining both the micro process execution and program slicing metrics. Bugs were grouped according to their bug fixing process execution against both Bug CC and Bug LoC. These results suggest that bugs with incorrect sequence in their bug fixing processes show a wide distribution with very high cyclomatic complexity and lines of code as compared to bugs that follow the general bug fixing model and bugs that are closed in the same day as being fixed. This data however may not be reliable as the sample set for incorrect sequence is too small to have any statistical significance.

4.2 Testing Hypothesis

Putting together the bug classifications and the analysis of the micro process as a proof of concept, there is enough evidence to suggest that there is indeed a relationship between bug characteristics and its processes executed. For example for bugs taking up to 10 days to fixing generally have low complexity and lines of code show a tendency to closed in the same day and have a higher likelihood to have errors in its process execution.

In response to the hypotheses, the findings support Hypothesis 1 as our research was able to successfully classify bugs with similar micro process executions and have a particular program slicing metric. For example, bugs with incorrect micro process execution show much higher Bug CC metric

than the correct standard and fixed and closed in same day bugs. Hypothesis 2 is also supported by the evidence that higher Bug CC is more prone to incorrect sequences. However, further research it needed to have more confidence in these hypotheses.

4.3 Application of Research

The results of the research act as a proof of concept for the use of program slicing in the inspection of the bug fixing processes in a software development project. It is envisioned that the research will help contribute to a better understanding and classification of bugs based on the nature of code.

If successful, the research can used to help create ‘prediction models’ for bugs. For example, based on code, a bug could be classified, thus assist developers handle the bug faster. This would then contribute to software improvement at this micro level.
The implementation could be a tool that is used during the bug detection and assignment stage of the bug fixing process. It would be able to analyze what part of the code are affected, and based on the classification, predict how long the bug would normally take to fix as well code based metrics such as the complexity of code.

4.4 Future Works

As mentioned throughout the paper, this work is seen as proof of concept with the final aim of developing a prediction model for bug fixing process based on the bug’s characteristics. Future work will be to refine the tools and metrics used. Currently for this research, only two program slicing metrics were introduced. As indicated by the results, extensive analysis is needed to make more concrete judgments.

More advance program slicing metrics such as lines of code affected in forward slicing and backward slicing will be explored. This a more precise metric, than the lines of code used in this study. It is envisioned that this will give more precise analysis.

Lastly, a larger sample data will be experimented to further validate the results. Current work only shows three projects. It would be preferable to test more projects.

5. Conclusion

The results indicate that our alternative assessment approach for Software Process is feasible as a proof of concept. The study serves as the initial step for assessment of software process by analysis of micro processes.

Acknowledgements

This work is being conducted as a part of the StagE project, The Development of Next-Generation IT Infrastructure, supported by the Ministry of Education, Culture, Sports, Science and Technology.

References

[1]. J. Herbsleb, A. Carleton, J. Rozum, J. Siegel and D. Zubrow, "Benefits of CMM-Based Software Process Improvement: Initial Results", (CMU/SEI-94-TR-13). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1994.

[2]. K. K. Margaret and J. A. Kent, "Interpreting the CMMI: A Process Improvement Approach", Auerbach Publications, (CSUE Body of Knowledge area: Software Quality Management), 2003.

[3]. C. H. Schmauch, "ISO 9000 for Software Developers", 2nd. ASQ Quality Press, 1995.

[4]. S. Beecham, T. Hall and A. Rainer, "Software process problems in twelve software companies: an empirical analysis", Empirical Software Engineering, v.8, pp. 7-42, 2003.

[5]. N. Baddoo and T. Hall, "De-Motivators of software process improvement: an analysis of practitioner's views", Journal of Systems and Software, v.66, n.1, pp. 23-33, 2003.

[6]. M. Niazi, M. A. Babar, "De-motivators for software process improvement: an analysis of Vietnamese practitioners' views", in: International Conference on Product Focused Software Process Improvement PROFES 2007, LNCS, v.4589, pp. 118-131, 2007.

[7]. J.G. Brodman and D.L. Johnson, "What small businesses and small organizations say about the CMMI", In Proceedings of the 16th International Conference on Software Engineering (ICSE), IEEE Computer Society, 1994.

[8]. A. Rainer and T. Hall, "Key success factors for implementing software process improvement: a maturity-based analysis", Journal of Systems and Software v.62, n.2, pp.71-84, 2002.

[9]. C. Yoo, J. Yoon, B. Lee, C. Lee, J. Lee, S. Hyun and C. Wu, "A unified model for the implementation of both ISO 9001:2000 and CMMI by ISO-certified organizations", The Journal of Systems and Software 79, n.7, pp. 954-961, 2006.

[10]. O. Armbrust, M. Katahira, Y. Miyamoto, J. M?nch, H. Nakao, and A. O. Campo, "Scoping software process lines", Softw. Process , v.14, n.3, pp.181-197, May, 2009.

[11]. S. Morisaki and H. Iida, "Fine-Grained Software Process Analysis to Ongoing Distributed Software Development", 1st Workshop on Measurement-based Cockpits for Distributed Software and Systems Engineering Projects (SOFTPIT 2007), pp.26-30, Munich, Germany, Aug., 2007.

[12]. M. Weiser, "Program slicing", In Proceedings of the 5th international Conference on Software Engineering (San Diego, California, United States). International Conference on Software Engineering. IEEE Press, Piscataway, NJ, pp.439-449, Mar. 09 - 12, 1981.

[13]. T. Hall, P. Wernick, "Program Slicing Metrics and Evolvability: an Initial Study ", Software Evolvability, IEEE International Workshop, pp.35-40, 2005.

[14]. K. Pan, S. Kim, E. J. Whitehead, Jr., "Bug Classification Using Program Slicing Metrics", Source Code Analysis and Manipulation, IEEE International Workshop, pp.31-42, 2006

[15]. A. Watson and T McCabe, "Structured testing: A testing methodology using the cyclomatic complexity metric", National Institute of Standards and Technology, Gaithersburg, MD, (NIST) Special Publication, pp.500-235, 1996.

[16]. B. Xu, , J. Qian, X Zhang, Z Wu, and L. Chen, "A brief survey of program slicing", SIGSOFT Softw. Eng. Notes v.30, n.2, pp.1-36, Mar., 2005.

[17]. V. P. Ranganath , J. Hatcliff, "Slicing concurrent Java programs using Indus and Kaveri", International Journal on Software Tools for Technology Transfer (STTT), v.9 n.5, pp.489-504, Oct., 2007

[18]. T. Wang and A. Roychoudhury, "Dynamic slicing on Java bytecode traces", ACM Trans. Program. Lang. Syst. v. 30, n.2 pp.1-49, Mar., 2008.

[19]. P. Anderson , M. Zarins, "The CodeSurfer Software Understanding Platform", Proceedings of the 13th International Workshop on Program Comprehension, pp.147-148, May 15-16, 2005

[20]. J. Bevan, E. J. Whitehead, S. Kim and M. Godfrey, "Facilitating software evolution research with Kenyon", In Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT international Symposium on Foundations of Software Engineering (Lisbon, Portugal). ESEC/FSE-13. ACM, New York, pp. 177-186, Sept. 05-09, 2005.

[21]. J. Howison and K. Crowston, "The perils and pitfalls of mining SourceForge", presented at Mining Software Repositories Workshop, International Conference on Software Engineering , Edinburgh, Scotland, May 25, 2004.