Software Defect Taxonomy, Analysis and Overview

Ali A Karahroudy, M.H.N. Tabrizi
Asgharykarahroudy10@students.ecu.edu, Tabrizim@ecu.edu
Computer Science Department, East Carolina University, Greenville, NC, USA

Abstract - *In this paper an overall analysis of current defect taxonomies is presented also plans for well defined process based taxonomy is carefully created using the existing models. The existing software defect taxonomies do not focus fully on the process, in most cases process and product are studied in parallel and significant amount of time is spent to identify and debug the defects or prevent them from occurring again. Our study is focused on the defects found based on the process in which they are found and selected based on the largest potential impact on the final product that will have significant impact in defect prevention.*

Keywords: Defect, Taxonomy, Process, Model, Frame ware, Attribute

1. Introduction

Various studies attempt to quantify the losses caused by software defects. As a recent instance Toyota has announced a recall for its 2010 hybrid vehicles. A break problem caused by faulty antilock braking software is believed to be the reason [1]. Tens of thousands of medical devices were recalled in March 2007 to correct a software bug [2]. Another example is the first flight of the European Space Agency's new Ariane 5 rocket failed shortly after launching [3], resulting in an estimated uninsured loss of a half billion dollars. It was reportedly due to the lack of exception handling of a floating-point error in a conversion from a 64-bit integer to a 16-bit signed integer.

It is also reported that [4] software bugs caused the bank accounts of 823 customers of a major U.S. bank to be credited with $924,844,208.32 each in May 1996, according to newspaper reports. The American Bankers Association claimed it was the largest such error in banking history. According to bank spokesman the programming errors were corrected and all funds were recovered. One study [5] estimated losses to the U.S. economy alone due to software defects at $60 billion annually.

There are many instances of monumental software failures that have staggering losses up to and including the loss of human life. In 2010, a software problem caused bank cards to fail across Germany [6]**.** It is also likely that Air France flight 447 was brought down due to software defect that was not able to handle the extreme weather conditions. In 1999, the Mars Climate Orbiter was lost due to software confusing pounds with kilograms. One of the most notable instances of software failure was the Therac-25 radiation treatment system [7]. In this case, faulty software caused patients to be given massive overdoses of radiation, killing several of them.

Defect prevention in early stages of software development life cycle is very cost effective. Meanwhile the more defect prevention processes are in place, the more costs will be imposed to the project budget therefore a very well balanced model is needed to fulfill both defect prevention and cost effectiveness.

However Software's complexity and accelerated development schedules make defects prevention difficult. Also current models like peer reviews, analysis tools, and different testing techniques detect different classes of defects at different points in the development cycle. [8] providing the justification that no existing model can be enough as a standalone model.

Organizations are still asking how they can predict the quality of their software despite the substantial research effort over the last 30 years. [9] where wide range of prediction models have been proposed. Complexity and size metrics have been used in an attempt to predict the number of defects a system will reveal in operation or testing. Reliability models have been developed to predict failure rates based on the expected operational usage profile of the system. The maturity of design and testing processes has been

advanced as ways of reducing defects. Recently large complex multivariate statistical models have been proposed in an attempt to find a single complexity metric that will account for defects [9].

1. Software Defect Taxonomy

It is reported that the best way to prevent and control software defects is using proper defect taxonomy [10] (A defect is a structural property of software document of any kind, namely a deviation from the nearest correct document that makes the document incorrect or locally incorrect. Taxonomy is a system of hierarchical categories designed to be useful aid for reproducibly classifying things) the area of software quality measurements and quantification is beset with undue complexity and has, in some ways, advanced away from the developer [11]. In an area where the processes are so amorphous, the tangibles required for measurement and modeling are few. With the result academic pursuits that can't be confined to the limitations of practice evolved and became distanced from the developer. In this area, the need to derive tractable measurements that are reasonable to undertake and intuitively plausible cannot be understated. Measurement without an underlying theme can leave the experimentalist, the theorist and the practitioner very confused.

Defect removal and defect prevention techniques [12] are no longer good enough to inspire confidence for software products. Techniques that help predict the number of remaining defects in software products can further boost customer confidence. Such techniques are easy to perform and have been used outside the realm of software engineering to produce reliable estimates for decades in the area of animal, bird, fish, and insect counts, and more recently for estimating the prevalence of “SARS” (Severe Acute Respiratory Syndrome) and cancer occurrences.

In this paper we will review major taxonomies that are being used by software developers.

1.1. Orthogonal Defect Classification

The ODC (Orthogonal defect classification) [13] is a scheme to capture the semantics of each software defect quickly. It is the definition and capture of defect attributes that make mathematical analysis and modeling possible. Analysis of ODC data provides a valuable diagnostics method for evaluating the various phases of the software life cycle (design, development, test and service) and the maturity of the product. ODC makes it possible to push the understanding and use of defects well beyond its quality.

An evolved model based on ODC is introduced by Madachy and Bohme [14] is called Orthogonal Defect Classification COnstructive QUALity Model (ODC COQUALMO) that predicts defects introduced and removed, classified by ODC types. Using parametric cost and defect removal inputs, static and dynamic versions of the model help one determine the impacts of quality strategies on defect profiles, cost and risk.

The dynamic version provides insight into time trends and is suitable for continuous usage on a project. The models are calibrated with empirical data on defect distributions, introduction and removal rates; and supplemented with Delphi results for detailed ODC defect detection efficiencies. This work has supported the development of software risk advisory tools for NASA flight projects. They have demonstrated the integration of ODC COQUALMO with automated risk minimization methods to design higher value quality processes, in shorter time and with fewer resources, to meet stringent quality goals on projects. There are different implementations of ODC COQUALMO as static versions in a spreadsheet and one that runs on the Internet that estimate the final levels of defects for the ODC categories. They have developed a dynamic tool to apply COQUALMO in the field that could be found at [15].Different methods for risk analysis and reduction have been performed in conjunction with ODC COQUALMO, which can produce optimal results in less time

Another technique to reduce risks with the model is a strategic method of optimization. It generates optimal risk reduction strategies for defect removal for a given budget, and also computes the best order of activities. An integration of ODC COQUALMO has also been prototyped with the DDP risk management tool which uses fault trees to represent the overall system's dependencies on software functionality. These experiments to optimize quality processes are described in more detail in [16].

1.2. Defect Severity and Defect Priority

Based on [17] the severity framework for assigning defect criticality that has proven that a five level criticality scale is the most effective scale to study defects. The criticality associated with each level is based on the answers to several questions:

- It must be determined if the defect resulted in a system failure.
- The probability of failure recovery must be determined.
- It must be determined if the system can do this on its own or if remedial measures must be implemented in order to return the system to reliable operation.
- It must be determined if the system can operate reliably with the defect present if it is not manifested as a failure.
- It must be determined if the defect should or should not be repaired.

The five level scale of defect criticality addresses the above mentioned questions are; critical, major, average, minor and exception. In addition to the defect severity level defined above, defect priority level can be used with severity categories to determine the immediacy of repair. A five repair priority scale has also been used in common testing practice. The levels are: resolve immediately, give high attention, normal queue, low priority, and defer.

1.3. Statistical Defect Models

The goal of statistical defect modeling, which includes what is commonly referred to as *Software Reliability Growth* [18], has been to predict the reliability of a software product. Typically, this may be measured in terms of the number of defects remaining in the field, the failure rate of the product, the short term defect detection rate, etc. Although this may provide a good report card, it often occurs so late in the development cycle that makes it of little value to the developer. Ideally, a developer would like to get feedback during the development life cycle.

1.4. Qualitative Casual Analysis

To identify the root causes of the defects, the defects are analyzed, one at a time, by a team that is knowledgeable in the area.

At IBM, the Defect Prevention Process and similar efforts elsewhere have found causal analysis to be very effective in reducing the number of errors committed in a software project [19]. The qualitative analysis provides feedback to developers that eventually improve both the quality and the productivity of the software organization. Defect prevention can provide feedback to developers at any stage of their software development process.

However, the resources required to administer this method are significant, although the rewards have proven to be valuable. Moreover, given the qualitative nature of the analysis, the method does not lend itself well to measurement and quantitative analysis. Consequently, defect prevention, though not a part of the engineering process control model, could eventually work in conjunction with it.

1.5. Peer Review Technique

Peer reviews, in particular software inspections, have become accepted within the software industry as a cost effective way of removing defects as early as possible in the software development cycle [20] The peer review process is also quite easy to measure.

Peer reviews are included in the Software Engineering Institute's Capability Maturity Model Integration (CMMI ®) [21] as a required process for those organizations following the CMMI as a guide for process improvement. Peer reviews are especially valuable as a way to remove defects early in the development cycle and should be performed on all major software development work products including requirements, design, code, and test procedures. The software inspection method of performing peer reviews is generally considered the most effective method of performing peer reviews [22] and is an indisputable software engineering best practice. Other types of peer reviews that are practiced with varying degrees of formality are team reviews, walkthroughs, and pair programming. Once peer reviews are an established practice, the data from each peer review can be used for defect management. For this purpose the following data from each peer review are recommended to be collected [23]:

- Program, function, and work product identifiers
- Type and phase of review, e.g., design walkthrough or code inspection
- Who attended and how much time was spent preparing for the review meeting
- How long the review meeting(s) lasted and who attended the meeting
- Size of the work product, e.g., pages of design or source lines of code (SLOC)
- Total major defects and total minor defects detected

For each defect found the following data is recommended [24]:

- Defect type, e.g., missing, wrong, or extra.
- Defect origin, i.e., the development phase where the defect originated, e.g., requirements, design or code. Note that it is possible that a defect can be discovered in a later phase than when it was first inserted, e.g., a design defect can be discovered during a peer review of code.
- Defect severity, i.e., major or minor. A major defect being any defect that could be discovered or observed by a customer during operational use of the software, or a defect that requires significant time to fix. Minor defects are everything else, e.g., documentation errors.
- Defect location, e.g., module or program element name.

And finally from the collected data the following derived measurements are recommended for each peer review:

- (Major) defects per detection hour – the average number of major defects found for each hour of detection time derived by dividing the number of major defects found by the sum of the participants' total preparation time and total time (labor hours) spent in the review meeting
- Average preparation rate - the average rate at which each reviewer prepared, e.g., pages per hour, which indicates how quickly the material was reviewed.
- Meeting rate, e.g., pages per hour
- (Major) defect detection rate – the ratio of the number of defects found per the size of the work product, e.g., defects per page or defects per 1000 SLOC (KSLOC)

After a sufficient amount of peer review data has been collected by similar projects and data integrity has been established, average rates can be established for the 3 preparation, meeting, and defect detection rates (for each type of peer review for each type of work product). This is usually done by the organization's measurement guru or analyst. From these averages high and low detection rate thresholds can be established to trigger further analysis of the data and possible action. An advanced method is to apply statistical process control (SPC) [25] and calculate the normal range of performance for each of these rates.

2. Process Based Defect Taxonomy

Process based taxonomy assumes that defects are recorded when they are found throughout the software process lifecycle, including their classification according to the defect taxonomy. Recording defects and classifying them can help understand the process with respect to all those activities that produce defects in particular, they help in identifying process weaknesses (high-defect steps). In the other view, after product testing a list of defects is produced that pinpoints the defects in product and their roots causes. This view is not talking about the process as a root to defect and usually brings up suggestions for rework approach and believes that the kinds of defects may point out the best approach for doing rework (e.g. direct repair, review, redesign, retest, etc.). Defect characteristics of artifacts may help with risk assessment for process decisions such as task priorities, go/no-go decisions, reimplementation decisions etc.

Despite high amount of effort into studying root cause analysis or qualitative analysis and bringing up "Why" a specific defect is "produced" and how to prevent it in future products, in most of the studies the importance of process has been ignored. if a taxonomy comes up with processes as center of gravity, and point out the probability of preventing defects based on process developments, that not only will help the future projects – as the current process based taxonomies do – but also serves to make a better final product for an ongoing project.

2.1. Defect Driven Process Taxonomy

One of the biggest contributors to software defect is "human" factor that is usually underestimated by others. It seems like that they focused on the results of human actions rather than "nature" of human behaviors that will lead to those results.

A study is needed with focus on defects based on the process in which they are found. These defects are selected based on their potential impact on the final product.

This study should focus on the following characteristics of the process:

- When process starts and when ends
- What are the major and minor goals of the process
- How human behaviors can affect the process and in which level
- Is it possible to break down the process to sub-processes

- Define the defects that can be produced by specific process
- Define the severity of those defects and their affect in project success
- Use mathematical methods for modeling the process

Then a process improvement method can be elicited from this study. The mathematical methods will then be deployed to help in quantifying the processes in order to make them more manageable and understandable.

2.2. Framework

Generally to the authors believe the process based taxonomy framework should focus on the following:

- Process Attributes;
  - Taxonomy is defined by attributes. Each attribute has a set of values. The values represent defect characteristics that must be registered at the process studying procedure.
  - These attributes should be the ones which can help analyze the process in future.

- Process Structure;
  - Structure defines the relationship between processes and the way they interact with each other.
  - One of the most used structures can be orthogonal relationship that has been vastly used by IBM.

- Process Properties;
  - Unique properties of each process that can be helpful for identifying that process as a "type" and be used in future cleaning procedures to make it faster and more effective should be defined for each process.

- Effectiveness rate;
  - A touchable parameter to evaluate each cleaned process is needed to be in place that will be used in order to compare the results.

Finally a modeling method can combine the entire information gathered through this procedure to introduce software defect taxonomy. This taxonomy could prove to be much more cost effective than those already exist.

Conclusion

Much of the published work in the defect Taxonomy modeling area is focused on debugging rather than defect prevention and mostly they emphasis the final product or testing results. This review paper shows that many past studies have suffered from including the process as a standalone artifact. The issues and problems surrounding the models illustrate how difficult defect prediction is and how easy it is introduce modeling errors. Specifically, we found out that those existing models using size and complexity metrics alone are incapable of predicting defects accurately. Furthermore, these models do not describe how defect introduction and detection variables affect defect quantity and quality. The existing software defect taxonomies often do not focus fully on the process based approach. In most cases process and product are studied in parallel and significant amount of time is spent to identify and debug the defects or prevent them from occurring again. To the author's believe, considering a process based taxonomy which is carefully created and implemented benefits from strength of already developed taxonomies and avoid their weaknesses.

References:

[1] Douglas A. McIntyre, 17 Feb 2010 *"24/7 Wall Street"*, http://247wallst.com/

[2] Wikipedia "List of Software Bugs", http://en.wikipedia.org/wiki/List_of_software_bugs

[3] Gleick James, A bug and a Crash, 1996 *"Sometimes a bug is more than a nuisance"*. http://www.around.com/ariane.html

[4] Softwareengineeringrefrence.com, Software Failure list, 2010 http://www.sereferences.com/software-failure-list.php

[5] National Institute of standards and technology NIST, WED, APR 28, 2010 14:59 http://www.itbusinessedge.com

[6] Byline Bill Hoffman, CTO, Kit ware, 2009 "Software defects, one developer at a time"

[7] Lim Joanne October 1998, "An Engineering Disaster: Therac-25" PP 1-2

[8] Boehm Barry, victor R. Basili , Jan 2001 "Software defect reduction top 10 List" http://portal.acm.org/citation.cfm?id=621640

[9] Norman E. Fenton, 1999, “A Critique of Software Defect Prediction Models”

[10] Freie Universitat Berlin Researchhome website https://www.inf.fu-berlin.de/w/SE/DefectTaxonomy

[11] Chillarege Ram, 1992, “Orthogonal Defect classification a concept for in-Process Measurement” http://www.chillarege.com/articles/odc-concept

[12] Joe Schofield from Sandia National Laboratories, 2005 , “Beyond Defect Removal: Latent Defect Estimation with Capture Recapture Method (CRM)”, PP 1-2

[13] IBM. Refer to IBM Center of software Engineering at IBM Research web site ODC. http://www.research.ibm.com/softeng/ODC/ODC.HTM

[14] Raymond Madachy and Barry Boehm from university of southern California, 2001, "Bayesian Analysis of Software Costs and Quality Models"

[15] http://csse.usc.edu/tools/odc_coqualmo.php

[16] Madachy R., Boehm B., Richardson J., Feather M., Menzies T.” Value-Based Design of Software V&V Processes for NASA Flight Projects, In: AIAA Space 2007 Conference, (2007)”, http://csse.usc.edu/csse/TECHRPTS/2008/usc-csse-2008-830/usc-csse-2008-830.pdf

[17] Pavankumar Pothuraju's weblog, 23 Aug 2007, “The library of software Testing” http://geekswithblogs.net/ppothuraju/Default.aspx

[18] Shaik Mohammad Rafi ,Dr. K. Nageswara Rao and Shaheda Akhtar, 2002, ” Software Reliability Growth Model with Logistic-Exponential Test-Effort Function and Analysis of Software Release Policy”. http://www.enggjournals.com/ijcse/doc/IJCSE10-02-02-50.pdf

[19] Chillarege Ram , 1993, “Process Measurement, Analysis and Control” http://www.chillarege.com/articles/odc-process-control

[20] Karl E. Wiegers, Peer Reviews in Software, Addison-Wesley, 2002, “Using Peer Review Data to Manage Software Defects” http://www.compaid.com/caiinternet/ezine/lett-defects.pdf

[21] Beth Chrissis Mary , et al., 2007 “CMMI ® for Development, Version 1.2, Addison Wesle”.

[22] Ronald A. Radice, “High Quality Low Cost Software Inspections, Paradoxicon Publishing”, 2002

[23]H. Lett Steven, 2002, “Using Peer Review Data to Manage Software Defects” http://www.compaid.com/caiinternet/ezine/lett-defects.pdf

[24] H. Lett Steven 2001, “General Management of Software Defects” PP 4-5

[25] *Addison Wesley*, 1999, “ Measuring the Software Process” PP 3

[26] Hower Rick, 2007, “software QA and testing frequently asked questions” http://www.softwareqatest.com/qatfaq1.html

[27] Raymond Madachy, Barry Boehm, USC-CSSE-2008-817, "ODC COQUALMO - A Software Defect Introduction and Removal Model using Orthogonal Defect Classification” ,