A Holistic Approach to Software Defect Analysis and Management

Ghulam Mujtaba, Dr. Tariq Mahmood, Professor Zafar Nasir

Sukkur Institute of Business Administration, Sukkur, Pakistan.
National University of Computer and Emerging Sciences, Karachi, Pakistan.

Abstract: During the execution of a given software development process, the corresponding software system can generate diverse types of defects. The process of fixing these defects occurs throughout the software development process. Generally, the process of defect fixation often creates new defects in software systems. Hence, previous “good” fixes can become current “bad” ones. In software projects, the number of defects can be considerably large, and defect management can become even more challenging when the development projects are large in size. Often in such projects, defects become latent defects and fixes become delinquent fixes. Moreover, bad fixes may cause the injection of new defects in the software. To enhance the availability, reliability and quality of software systems, it is important for software producers to increase the quality, efficiency and effectiveness of the defect fixation process. In this paper, we have introduced a software maintenance metric called Defect Fixation Efficiency (abbreviated as DFE) to quantitatively measure the effectiveness of a defect fixation process. We have also made an effort to determine the relationship of the DFE metric with delinquent fixes, customer satisfaction and the maintenance phase. This relationship implies that an effective and efficient fixation of defects leads to the injection of fewer defects in the software, and to an increased satisfaction of the customers. Finally we have devised a “2 × 2” matrix that acts as an indicator of software quality and of the quality of the execution of the software development process.

Key words:

Background and Motivation:

Computers and software systems are becoming omnipresent in these days. In the past, computers were used for mathematical calculation only (Peter Norton, 2006). But in the modern times, almost every business relies on computers, network and the Internet to accomplish their needs. For instance, amazon.com is a web application for selling the all types of books online. Banks need financial software systems to maintain the information of its account holders and to record their daily transactions. A retail store needs an inventory control software for maintain stocks, and for making invoices of customers. That is why it is important for the software to function properly over a long period time without generating any defect(s) (Stephen and Kan, 2003). Such high quality requirements are satisfied by the people, i.e., the so-called software developers or producers involved in software development with the use of variety of quality assurance activities, metrics and measurements. Also, customers’ satisfaction is the ultimate validation of software quality. In these days, many software systems are larger in size and complex containing millions of lines of code. Such systems typically involve hundreds of people in development process. In such systems it is practically infeasible to achieve defect free software. Consequently, various SQA (Software Quality Assurance) activities can be carried out to eliminate certain defects and hence make high quality software systems (Stephen and Kan, 2003). We now discuss some of these activities.

Software Quality Assurance Activities:

To develop a quality software product, we need to execute the SQA activities during the development of software product. These activities guarantee that there are no or very few defects remaining in the software when it is shipped to customers or released in the market. SQA combines together various tasks and activities such as Formal Specification, Formal Verification, Formal Technical Reviews, Inspection, Testing, etc to produce high quality software. These terms are discussed in the next paragraph.

In (Jeff Tian, 2005), the authors have defined a classification scheme for SQA activities. There are three main classes of SQA activities i.e. Defect prevention, Defect reduction and Defect containment. This classification scheme is summarized in Figure 1. The basic purpose of defect prevention is error source blocking by performing activities like education and training, formal specification, formal verification and proper selection of tool, process, technologies and standards. Formal specification is responsible for producing the unambiguous set of
software requirement specification. Formal verification verifies the correctness of given software against its formal specifications (Stephen and Kan, 2003; Jeff Tian, 2005). The Defect reduction usually deals with the elimination of faults from software programs. Defect reduction activities include inspection, Formal Technical Reviews, Checklists, Inspections, Walkthroughs and testing. The objective of formal technical reviews is to uncover the errors in software and to verify that the software which is being under reviewed meets its requirements.

Moreover, a checklist is the list of tasks to be checked or reviewed. Inspection is the formal way to examine the software artifacts, identifying faults from those artifacts and removing those faults. Walkthroughs are the special from of reviews. These are more organized and usually applied to design and code. Testing is used on software systems to examine that software systems would work as expectedly their target customers. The Defect containment usually avoids global failure in presence of local failure through fault tolerance and safety assurance. Fault tolerance enables the system to operate properly in the event of failure. For fault tolerance, recovery blocks and N-version programming techniques are used. Recovery blocks use the checkpoints and repeat some computation steps in case of any fault or failure. N-version programming avoids software faults by running duplicate copies of software itself. In safety assurance our primary goal is to prevent the accidents and secondary goal is to reduce the damage of accident if failure occurs. For safety assurance, hazard elimination, reduction and control techniques are used (Stephen and Kan, 2003; Jeff Tian, 2005). The applicability of such activities will increase the software quality & reliability and reduce the chances of software failure and hence increase the chances of customer satisfaction. Generally, SQA activities start parallel with the software development process phases. This enables the analyst to achieve a high quality of specification and design. We will now discuss about metrics for measuring software quality, in the context of this paper.

**Software Quality Metrics:**

Quantitative measurements are very important in all domains including that of software development. Software metric is a measure of some property of a piece of software or its specification. There are different categories of software metrics such as product metrics, process metrics, project metrics, quality metrics, etc. Product metrics describe the characteristics of the product such as complexity, design features, size, performance, etc. Process metrics are used to improve the software development and maintenance such as defect arrival pattern, defect removal effectiveness, etc. Project metrics describe the project characteristics and execution, such as staffing pattern, cost, schedule, etc. Software quality metrics are responsible for measuring the quality aspects of the product as well as process. There are three sub types of software quality metrics i.e. end product quality metrics, in-process quality metrics and maintenance metrics (Stephen and Kan, 2003). Figure 2 shows the summarized classification of software quality metrics.

**End Product Quality Metrics** are used to measure the quality of the end product. Examples of such metrics include MTTF (mean time to failure), defect density (lines of code and functional point), customer reported problems (problem per user month) and customer satisfaction (percent satisfied). MTTF measures the time between failures. It is usually used in safety critical systems such as fire alarm systems. The defect density measures the number of defects relative to the software size. Customer reported problems or problems per user
month (PUM) metric computes the problems, users faced during the usage of software product. For measuring the customer satisfaction, percent satisfied metric is used, which calculates the percentage of customer satisfaction towards the use of specific software product. In Process Quality Metrics tracks pattern of defect arrival during testing. The examples are defect density during formal machine testing, defect arrival Pattern during formal machine testing, DRE (defect removal efficiency) and phase based defect removal pattern (Ayewah and N. Pugh, 2009; Alshathry et al., 2009; Korhonen and Salo, 2008). Defect density and defect arrival pattern during testing usually computes the number of defects during the formal machine testing.

Defect removal efficiency metric measures the test effectiveness. Phase based defect removal pattern measures the number of defects in software development phases such as design reviews, code inspection, etc. Software maintenance metrics such as Fix backlog and backlog management index, Fix response time and Percent delinquent fixes. Fix backlog is concerned to defects arrival rate and the time at which the fixes for reported problems become available. Fix response time is the mean time of all problems form open to closed fixes. Percent delinquent fixes calculate percentage of delinquent fixes (Fixes that exceed the response time criteria by severity level) (Stephen and Kan, 2003).

In this paper we have successfully introduced defect fixation efficiency metric in the domain of software maintenance. This metric will measure the effectiveness of defect fixation process by dividing the number of defects fixed with total number of defects plus new injected defects. We have also devised a “2 X 2” matrix that acts as an indicator of software quality and of the quality of the execution of the software development process. The matrix includes the SQA activities efforts and quality of defect fixation process. In section-II of this paper we have discussed the related work. In section-III we have discussed defect fixation efficiency metric. In section-IV we have discussed the “2X2” matrix. In section-V we have discussed the conclusion and future work.

Related Work:
Software quality plays a vital role in the success or failure of software projects. The absence of quality attributes and presence of defects in the software systems minimizes our business, and makes our customers dissatisfied. And according to (Pete Babich, 1992) usually the satisfied customers continue to purchase products from the same company, where dissatisfied customers start purchasing from some other company. So, it is essential for software organizations to make high quality software and focusing on the key customer quality requirements i.e. Usability, Portability, Testability, Reliability, Accuracy, etc. According to (Ramón Silva Leite, 2009) software quality enhances the customer satisfaction and customer loyalty. So, we can infer that software quality is directly proportional to customer satisfaction and customer satisfaction is directly proportional to customer loyalty. That’s why lots of research work is going on software quality. In (Stephen and Kan, 2003; Jeff Tian, 2005; Alshathry et al., 2009) there have been discussed various software quality assurance activities such as defect prevention activities, defect reduction activities and defect containment activities and various software quality metrics such as MTTF, DRE, Defect density, Phase based defect removal and software maintenance metrics.

Generally, software systems contain defects. In the software industry, the cost of fixing quality defects goes
up dramatically during the back-end activities, or the latter phases of the software development process. Figure 3 shows the relationship of development and relative cost to correct a defect. The x-axis of the figure contains the software development process phases i.e. requirement, design, code, test and operation and y-axis contains the relative cost to correct a defect. From this figure, we can infer that to correct a defect at initial phases such as requirement or design will result in low cost as compared to remove the defects at later phases such as testing. From this figure we can also infer that the cost can be saved simply by taking corrective actions in the front-end activities of software development process i.e. requirement or design phase. Also, according to (http://www.idc.com), software maintenance costs two-third of the overall cost of software development. According to (Collofello and Woodfield, 1989) software developers spend approximately 50 to 80% time in understanding and fixing the defects. It is not always correct that fixing a defect will improve the software quality.

![Fig. 3: Cost of Change Curve](image)

A bad fix also decreases the quality of software system. Thus, detection and correction of bad fixes earlier will enhance the quality and reliability of software system. According to (Zhongxian Gu, et al., 2010), the reopened bugs indicate the bad fixes. Reopened bug is the bug that has gone through the entire bug life cycle (discovered, reported, fixed) but upon verification was found to still be a problem. Approximately 38% to 50% bugs reopened due to bad fixes. Moreover, in (Weiss et al., 2007), an approach has been introduced for early effort estimation of fixing the defects, which helps in assigning issues and scheduling stable software releases. This approach has been evaluated on the JBoss project data and found that they can estimate within seven hours of the actual effort. There approach shows that whenever a new bug or defect comes in software system a new problem report has been generated. Then from bug database the most similar recorded bug effort is being calculated and from that efforts can be predicted.

Research in defect fixation normally falls into two categories i.e. empirical study and mining software repositories. BugMem (Kim et al., 2006) mines defect fixation history to predict potential bugs. In his work he devised an algorithm for bug finding by using the history or records of fixing the bugs. The algorithm finds the bugs and suggests the related fixes. In (Slawierski, et al., 2005), the authors propose a way to locate code fixes using repository mining. In their work they use a project bug database and project version archive to identify the changes that caused the problems. Code change usually occurs in software development. In (Ryder and F. Tip, 2001; Wloka et al., 2009) authors have proposed change impact analysis to find failure in code change and thus judge the change quality. Differential symbolic execution (Person et al., 2008) finds the difference between two versions of same program.

Software maintainability is one important aspect in software product. It is difficult to accurately predict the cost and risk of maintenance after the shipment or release of software products. A software maintainability prediction allows organizations to forecast the maintainability of their software systems to better manage their maintenance resources. This will help in reducing the maintenance effort and the overall cost of the software systems. In this context, in (Liang Ping, 2010) HMM (Hidden Markov Model) has been used to stimulate the maintainability of the software systems. In this HMM, health index has been used as a weight on the process of maintenance behavior over time. When the occurrence probabilities of maintenance behaviors reach a certain (pre-determined) count, that particular instant is the indication of a decline in the status of a software product. In fact, longer the time better is the maintainability. Moreover, in (Cong Jin Jin-An Liu, 2010), the techniques of Support Vector Machines (SVM) and clustering have been used in software maintainability prediction using object-oriented metrics. In this work, “maintenance effort” has been used as a dependent variables and “object oriented metrics” as the independent variable. The result is that both SVMs and clustering
techniques were very useful in constructing software maintainability prediction. As we can infer from the related work, a lot of research work is going on in software maintenance and defect fixation. Our approach is different, because in this research we have introduced software maintenance metric i.e. DFE or Defect Fixation Efficiency, which will calculate the effectiveness of defect fixation. We have also shown the relationship between the SQA activities’ efforts and the quality of the fixes.

**Defect Fixation Efficiency:**

In this section, novel software maintenance metric has been introduced, labeled as DFE (Defect Fixation Efficiency). To define defect fixation efficiency clearly, we must first understand the activities in the development process that are related to defect injection and removal. Normally, the phases in the software development process are requirement phase, high level design phase (I0), low level design phase (I1), code implementation phase (I2), unit test phase, component test phase, integration test phase and system test phase. Requirement phase, high level design phase, low level design phase, code implementation phase and unit test phase are referred as the front end activities of software development process. Component test phase, integration test phase and system test phase are referred as the back end activities of the software development process. For instance the defect injection activities in requirement phase are requirement gathering process and development of programming functional specifications. Defect removal activities in requirement phase are requirement analysis and requirement review.

**Table 1: Phase wise defect injection and activities**

<table>
<thead>
<tr>
<th>Development Phase</th>
<th>Defect Injection</th>
<th>Defect Removal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Requirement gathering process, Functional Specifications</td>
<td>Requirement analysis and review</td>
</tr>
<tr>
<td>High Level Design</td>
<td>Design work</td>
<td>High level design inspection</td>
</tr>
<tr>
<td>Low Level Design</td>
<td>Design work</td>
<td>low level design Inspection</td>
</tr>
<tr>
<td>Code Implementation</td>
<td>Coding</td>
<td>Code Inspection</td>
</tr>
<tr>
<td>Unit Test</td>
<td>Bad fixes</td>
<td>Testing</td>
</tr>
<tr>
<td>Component Test</td>
<td>Bad fixes</td>
<td>Testing</td>
</tr>
<tr>
<td>System Test</td>
<td>Bad fixes</td>
<td>Testing</td>
</tr>
</tbody>
</table>

Defect Fixation Efficiency can be defined as follows:

\[
\text{DFE} = \frac{\text{Total number of defects fixed}}{\text{Total number of defects} + \text{New injected defects}} \times 100\%
\]

The DFE can be calculated by dividing the number of defects fixed with the total number of defects plus new injected defects. For instance, let’s suppose the software contains ten (10) defects. Out of ten defects we fixed five (5) defects. And the fixing of these five (5) defects injected two (2) new defects. In this case the efficiency of defect fixation will be 41.6% (DFE = 5/(10+2) * 100). This metric can be calculated for the front end activities and for back end activities. The higher value of DFE metric shows the effectiveness of software development process and the software maintenance. The higher value of metric also shows that there are fewer defects that “escape” into the next phase, or into the field.

To derive an operational definition, we presented a matrix approach in Table 2 by cross classifying the defect data in terms of development phase in which the defects are found & removed and the phases in which the defects are injected. The data in Table 2 is taken from the IBM Rochester Project of AS/400. Table 2 shows that 124 defects exist in requirement phase. There exist 983 defects (859+124) in high level design phase and 731 defects were removed at high level design phase. There exist 1191 defects (939+859+124-731) in low level design phase. There exist 2000 defects (1538+939+859+124-731-729) defects in code implementation phase and 1097 defects removed at code implementation phase. Likewise that we can count in unit test, component test and system test. In unit test column the ‘2’, in component test column ‘4’ and in system test column ‘1’ represent the bad fixes. Using above data we can calculate the defect fixation efficiency in unit test, component test and system test by using our proposed DFE metric formula.

Using Table 2 data we can now calculate the defect fixation efficiency for each phase. The defect fixation efficiency for I0, I1, I2, unit test, component test and system test is given in Figure 4. Figure 5 shows the bar chart showing defect fixation efficiency of each phase of software development process.
The concept of defect fixation efficiency is central to software maintenance. Defect fixation is one of the top expenses in any software project and it greatly affects the customer satisfaction. After the completion of software development, maintenance phase starts when it is shipped to the customer end. During this phase, the customer problems arrive by time interval. The primary responsibility of the software developers is to fix the customer reported problems with excellent fix quality and without delinquency. Customers become dissatisfied when they encounter functional defects in the software when they run their business on the software. They become even more dissatisfied if the fixes of reported problems become faulty fixes and introduced new defects in software system. In software maintenance phase, the key quality goal is zero defective fixes without delinquency.

**Formation of “2×2” Matrix:**
In this paper, “2 × 2” matrix has been devised as an indicator of the quality of the software, and the quality of the process execution based on the SQA activities’ efforts and the quality of the fixed. In this context, following four scenarios has been identified: (1) High-High Scenario (2) High-Low Scenario (3) Low High Scenario (4) Low-Low Scenario. In following section we explain the each scenario in detail. Figure 4 shows the formation of “2×2” matrix.

**High-High Scenario:**
In a high-high scenario, the quality of the defect fixation process is high and that of the SQA activities’ effort is also high. If, in an overall development process, high quality QA activities’ efforts have been made and high quality defect fixation has occurred, then following will occur:
- High level of customer satisfaction
- Less rework
- There will not be any delinquent fixes
- There will not be any latent defects in software
- Less support required in software maintenance
- Software has been rigorously implemented and executed
- Software development process is very mature.

**High-Low Scenario:**
In a high-low scenario, the quality of the defect fixation process is high and the SQA activities’ effort is low. If, in an overall development process, low quality QA activities’ efforts have been made and high quality defect fixation has occurred, then following will occur:
- There will be less customer satisfaction,
- There will be more rework
- There may be delinquent fixes in software systems,
- There may be latent defects in the software systems software system has low quality
- More support required to maintain a software
- Software has not been rigorously implemented and executed

**Low-High Scenario:**
In a low-high scenario, the defect fixation quality is low and the SQA activities’ effort is high. If, in an overall development process, high quality QA activities efforts have been made and low quality defect fixation has been occurred, then following will occur:

---

**Figure 6- “2×2” Matrix**

![2x2 Matrix](image)
More rework required
More chances of delinquent fixes
More chances of latent defects
More support required
Software development process at the backend is not good.

Low-Low Scenario:
In a low-low scenario, the quality of the fix is low and also the SQA activities’ effort is low. If, in an overall development process, low quality QA activities’ efforts have been made and low quality defect fixation has been occurred then following will occur:
- There will be no customer satisfaction
- Lot of rework required
- More chances of delinquent fixes
- No software quality
- More support required
- Software is not rigorously implemented and executed
- There is no any development process.

Conclusions and Future Work:
In this research paper, novel software maintenance metric has been introduced labeled as Defect Fixation Efficiency metric, which is applicable to the domain of software maintenance. This metric measures the efficiency of defect fixation process. We have also formed a ‘2×2’ matrix to determine the relationship of defect fixation efficiency with the delinquent fixes, customer satisfaction and maintenance phase. Our work reveals that more effectively we fix the defect(s), lesser is the chance of injecting new defects in the software. Also, more effectively we fix the defect(s), the more is the chance for the reduction of the number of delinquent fixes. Moreover, more effectively we fix the defect(s), the more is the chance of increasing of number of satisfied customers. Thereby, we establish that the quality of execution of efforts has significant impact in improving the quality of defect fixes. In our future work, we will test the introduced metric on a live project in each phase of software development process i.e. from requirement phase to maintenance phase. We will also make a software prototype to automate defect fixation process and to enhance the efficiency of defect fixation efficiency metric.

ACKNOWLEDGEMENT
This is the extended version of our own paper presented and published as Conference proceedings in “International Conference on Computers & Emerging Technologies” (ICCET 2011) held on 22-23 April 2011 at Shah Abdul Latif University, Khairpur, Sindh, Pakistan.

REFERENCES


