# DEVELOPMENT AND EVALUATION OF A DEFECT TRACKING MODEL FOR CLASSIFYING THE INSERTED DEFECT DATA

***Prof.Torky Sultan***

***Ayman E.Khedr***

Information Systems Department Faculty of computers and Information,

Helwan University Cairo, Egypt

***Mostafa Sayed***

Information Systems Department Council state Cairo, Egypt

**Abstract**

Defect tracking systems play an important role in the software development organizations as they can store historical information about defects. There are many research in defect tracking models and systems to enhance their capabilities to be more specifically tracking. Furthermore, there are different studies in classifying bugs in a step by step method to have clear perception and applicable method in detecting such bugs. This paper shows a new proposed defect tracking model for the purpose of classifying the inserted defects reports in a step by step method for more enhancement of the software quality. Besides, an evaluation of experiment made for measuring the proposed factors results for defects classification.

**Keywords:**Bugs, Defects, Bug Tracking Systems, Defect Tracking Models, Software Quality

## Introduction

In many software development organizations, bug tracking systems play an important role as they allow different types of users communicating with each other (i.e. developers; testers and customers ) to assure that they have the same perception about problems or requesting new features. In addition, bug tracking systems can keep track of more historical information stored of the bugs. Earlier attempts were made for enhancing the defects tracking models and defect classifications (Just, 2008); (KO, 2003); (Curhan, 2005); (Edwards, 2006); (Endres, 1975) and (Janák, 2009). But there were no general farm work for concentrating on

different tracking phases. Besides, there were no interests with the insertion factors for classifying defects through those attempts. These issues make difficulty for retrieving accurate information from defects tracking tools in the future.

To address these issues, we have developed a general defects tracking model for classifying the inserted defect data in a step for more enhancing defects tracking model quality. In this paper we developed the model based on the previous works of (Edwards, 2006) and (Janák, 2009). Also we did evaluate the proposed model using an experiment. The model was helpful in describing and explaining different phases of defect tracking model, as well as the insertion factors for classifying defects.

This paper provides new proposed defects tracking model concentrating on the factors for the insertion of defects reports through tracking tools. In addition, it provides a theoretical overview of the literature on defects tracking systems and overview of different aspects of it and their components (section 2). The existing attempts to improve the defects tracking systems are highlighted in our synthesizing framework (section 3). Also, the paper provides a conceptual framework design for the proposed defects tracking model (section 4). The paper ends with an experiment for evaluating the proposed model (section 5), followed by section summary of (section 6).

**Overview**

This section aims at providing a detailed discussion of the background overviews about defect tracking systems.

There are many software tools that play an important role in tracking defects of software and which are called “Defects Tracking Systems”. Jalbert defined them as “Allow users to report, describe, track, classify and comment on bug reports and feature requests” (Jalbert, 2008).

Defects Tracking Systems can be separated systems that can integrate, and contribute in software development process. They can keep, with details of defects reports and information associated with resolving it, in a database storage. Lethbridge, Singer and Forward indicated that developers view the defect tracking systems as important repositories of historical information (Lethbridge, 2003). Furthermore, software defect data is an important source for the organizations for the software process improvement decisions and that “ignoring defect data, can lead to serious consequences for an organizational business” (Grady, 1996). In addition "they may be part of an integrated suite of configuration management tools, where the status of the defect may act as a trigger or key for other events within the system" (Avram, 2007).

There is no doubt that software quality which is used in detecting defects, is one of the important factors for evaluating the software process development. Weinberg (1983) documented that an error costing a company 1.6 billion dollars, and was the result of changing a single character in a line of code (Weinberg, 1983). Also, Curhan mentioned that "some types of defects have a much higher costs to fix due to the customer impact and the time needed to fix them, or wider distribution of the software in which they are embedded " (Curhan, 2005).

Moreover, a large number of software companies use Software Tracking Tools to achieve the goals of the Configuration Management. Janák defined configuration management as "the process of controlling and documenting changes to a developing system" (Janák, 2009). Also, software tracking tools help quality control engineers to accomplish their jobs as good as possible to discover, and prevent the occurrence of bugs by tracking them.

The Software Tracking Tools are simply built based on defect tracking models. Edwards and Steinke (2006) simply discussed the defects tracking model, as they divided it into the following two stages: ((repair /resolution) -(verification)) and the following three changes of status: (discovery – resolved – closed) (Edwards, 2006).

Microsoft Team Systems used a four-stage defects tracking model for Capability Maturity Model Integration (CMMI); the model expanded and evolved the "open" stage into the following two stages: "proposed" and "active" stage. Although the model enhanced the three- stage defects tracking model, it still works as a framework describing the status and phases of bugs that should follow. The three statuses (deferred – rejected – duplicate) duplicated through two positions, the proposed stage and the active stage (Microsoft, 2012). There were no remarks about how to examine and register the bugs.

Edwards et al. (2006), proposed the Full Product Life Cycle Defect (FPLC) Model, which was an extension of IBM/Rational Model with changes to include the test and project management interfaces. The model discussed in details, the five statuses of the defect tracking model which are: Submitted, Open, Postponed, Resolved and Closed. Although the model mentioned perfectly the duplication problem of defects; it still has some remarkable scope for more enhancements.

The research dealt with the status "reject" as not a closed status. It coped with it as a circulating process where it should be a “Closed” status. Also another remarkable note about the postponed defect, Edwards et al. (2006), reported that "Placing any defect in a Postponed status is a tacit admission that it should be repaired, but at a later time." (Edwards, 2006) which means that it has the priority to be repaired not to be a closed.

One of the famous defects tracking tools used by quality control engineers is a Bugzilla defect tracking system. The work flow of the model showed that it classified the new bugs into the following two categories: the first one comes from a user with a confirmation right, and the second comes from any user but it will not be confirmed till it has enough votes. Also, it concentrated on quality control engineer roles in checking the appropriate solution which being satisfied, verified, closed or didn't conform with the solution (Janák, 2009).

Although the default IBM Rational Clear Quest Ticket mentioned the workflow path that the defect process has taken, and which "Starts when the defect is discovered and ends when the defect is resolved, hopefully repaired, for the most immediate release of the software application" (Janák, 2009). It still has some shortcomings as the "rejected" status could be in any state. It may be after investigation, the approved state or after the task opened and in all the cases, it should be closed. Also the approved status should be one of the roles of quality control engineer; who should check it as the defect may not exist only in a new project process, but also may exist in the maintenance process.

**Synthesis Model for the Classification of the Bugs**

The last section discussed the different overviews of the defect tracking systems. Their workflow models, the status and paths of the defects through the process of discovering the defect. Also, it mentioned the literature reviews of different research at the same point that dealing with defects in their overall aspects. The proposed model concerned with the following two points: First the different classifications of the bugs; and the second is the different phases of tracking the defects. The two points will be discussed in more details in the next paragraphs.

This section covers discussing the proposed model and how it makes filtration and classification of the bugs. However, a bug in its default way, is discovered where an action or value is not achieved as it decided or going in an unexpected way. We divided the classification and track of the bugs into the following Phases: (Submission – Examination – Registration – Tracking) which interact with each other and will be discussed in more details later.

**The Submission Phase**

The first phase of the proposed model will help us in understanding the classification process of the bugs and the issues in the submission phase.

The role of this model in tracking bugs, starts after discovering an incorrect action or value to the system. Describing and stating the problem is a significant component for

retrieving a suitable solution. Stimson (1998) mentioned that "A problem well-stated is half-solved" (Stimson, 1998), this push us to define of who discovered the bug, and where it is discovered.

There are two different groups of users who can discover the presence bugs. The first Group is: "The Normal User" who deals with the system after released to him to achieve a specific function or goal. The second Group is: "The Authorized User" who participates at any phase of the development process. He may be one of triage team or development engineering team. The bug is usually discovered in two positions: the first position is through the development process. The second position is after releasing the product to users.

Section (2), discussed a number of different defect tracking models; where there were a number of these models which coped with "the submission phase" as the first step of filtration and classification defect reports. The adapted one with our model was "Bugzilla tickets workflow". We will modify it to be more compatible with the proposed model.

Bugzilla Workflow Model, classified the detectors of the bugs into two categories:-

1. Anyone who has enough votes.
2. A user with confirmation rights.

According to the classification of users in the Bugzilla Workflow Model, we will classify the users into three categories:-

1. Authorized user with confirmation rights.
2. Trusted User.
3. Normal User.

The first Category: (Authorized User) who is discovering the bugs inside the location of the development process. The authorized user may be one of the quality control engineers i.e. development engineer or may be anyone who has the ability to discover a bug. The second category: (Trusted User), who can be defined as the user who has the ability, and good experience in dealing with the product or system; also has a recorded history of detecting bugs. The third category: (Normal User) who has the ability but little experience of dealing with the product or system, and has a short recorded history of detecting bugs. That is, the Normal User has the ability to inform the presence of a bug but hasn't the priority element without a confirmation of an "Authorized User".

**The Examination Phase**

The examination phase begins after the end of the submission phase. In this phase, the outside or inside user who participates in the development process decides that there is a vision for a defect.

This phase has its own priority as Hooimeijer and Weimer (2007) documented that "Bug report triage and evaluation are the significant part of modern software engineering for many large projects"(Hooimeijer, 2007). It is the first phase of preventing the distortion of recording un-wanted data or duplication announcement of bugs by checking the database for recorded bugs before and through each time of registering a newly discovered bug.

According to the work and efforts made by Mays, Jones, Holloway and Studinski, at IBM 1990 for defects prevention. They analyzed the faults that appeared in order to understand them using casual analysis. In addition, detecting the way of prevents defects from appearing in the future. They showed the role and importance of the action team whose responsibility was to detect and store the appeared faults in the database and make a checklist to be updated with the new faults. Also the important role of "triage team", mentioned by Black (1999), who assured that the triage team can review, evaluate the defects and assigning them to the development team (Black, 1999). For more information in details about the triage team see (Mays, 1990).

When the Bug examination process is done, it is followed by rules and strategy of checking tests through quality control engineers. Furthermore, bug examination, is the last phase of deciding whether either the bug was recorded before with a suitable solution, or it will be a new classified bug.

The last statement leads us in the following three states after having the bug's examination recorded history such as:-

1. Bug not found and not registered before.
2. Bug found with the same condition and need to be in (reopened state).

The first point will be discussed in more details, in the next section as it will be the default path. This point was achieved by following different test case scenarios, and confirmation that there was a bug with the same conditions registered before. Therefore, a "reopen state" can be released by an authorized user in the examination phase in order to prevent duplication defect reports.

**The Registration Phase**

The registration phase follows the examination phase. It is an important phase for retrieving useful information in the future because there are different factors for classifying defects reports which were registered in this phase. The next paragraphs, will discuss different classification of defects schemes.

Although there is a large number of research on the classification of defects schemes, they faced a number of problems including ambiguous, and confusion of error causes (Ostrand, 1984).

One of the first works for classifying defects, was made by Endres in 1973 of IBM. He classified the defects into six general categories including: Machine Error, User Error, and Documentation Error. Also, he classified each defect by 'type'. But this type of classification scheme was very complex (Endres, 1975).

According to Basili, and Perricone's categories, the error classified as one of the following Categories: Requirements incorrect, Functional Specification misinterpreted, a design error which spans several modules, an implementation error in a single module, misunderstanding of the external environment, error in the use of the compiler, clerical error, and error due to previous wrong correction of an error (Basili, 1984).

Another work was made by Sullivan and Chillarege (1992) to analyze the different error classifications. They made their work based on defects reported at customer sites in two large IBM database management products, DB2 and IMS. They compared the error type; defects type and error triggers classifications (Tammana, 1998).

Fredericks and Basili (1998) made analysis to find defects and how organizations dealt with it. They focused on achieving three goals that can be defined as significance factors of building a new defect tracking models. These goals are: Detecting the Nature of Defects; Detecting the Location of Defects, and when the Defects are Inserted.

In the early 1990's, IBM developed two new Technologies using defects data. The First Technology: "Defect Prevention", which involves development teams contributing to a knowledge database containing: common defects, how they can be prevented, and how to easily detect them. The Second Technology: "Orthogonal Defect Classification", which involves using statistical methods to predict the phase in which a defect originated, rather than relying on the subjective evaluation of an engineer (Fredericks, 1998).

The Defect Tracking Model had evolved at the end of the nineties; the Defects Classification Scheme mentioned three Elements of defects Categorization. The First Element was the Location where the bug is discovered through the development process. The Second Element was the type of defect where may be classified through each phase of the development process has its own kind of defects. The Third Element was the value of each defect which can be measured (Pfleeger, 1998).

According to Rus (2002), there is a defect classifying schema that was developed and used by IBM called "Orthogonal Defect Classification". He defined it as "A measurement

concept for software development that uses the defect stream as a source of information on the product and the development process "(Rus, 2002). He divided it into two classes of defect attributes: the First Class associated with the defect discovery and contained the elements: (activity, trigger, and impact). The Second Class associated with the removal of the defect and contained the elements: (target, defect type, qualifier, source, and age).

With the last different views of the classification schemes of defects, it appeared that there were a number of factors that describe the defects, and these factors are so important. We will concentrate on the following two Factors that are seen in the degree of importance from a quality control perspective.

1. The First Element is "Bug Location":

The locations of the bugs are determined by, in which stage the bug appeared or discovered; and in which place in the system or the application it appeared. Fry and Weimer (2010) defined fault localization as: "Is the task of determining if a program or code fragment contains a defect, and if so, locating exactly where that defect resides" (Fry, 2010).

Furthermore, another element for describing the location of bugs is to describe where the bug was discovered through the system. Also we have to describe the surrounding environment of the system as an element in classifying the location of the bugs such the version of the system, the kind of operating system that the system works under.

2. The Second Element is "Bug Type":

The ‘Bug Type’ varies from one system to another because the different tools which were used to create such systems, have their own limitations and shortcomings according to the study made by Ko et al., (KO, 2003). However, we have to put a dynamic framework for defining the bug type according to the various tools used to build the systems; with respect to the major general bug type.

Sullivan et al., 1992 Classified the Software Defects Type as: function defect, data structure/algorithm defect, assignment/checking defect, interface defect, timing/synchronization defect, and build/package/merge defect (Sullivan, 1992). We proposed a general defect types which are Interface defect, calculation defect, loading defect, security defect, documentation defect, enhancement defect and business logic defect that may appear in any system.

**The Tracking Phase**

This phase is concerned with the traceability of the defects that were registered in the system before. There were a number of scenarios expected from the proposed model to

achieve. One of these scenarios is the traditional scenario which begins with a new classified defect on the status "Initial ".

At this stage, the development engineer who is responsible for fixing such defect starts to work under the status: "under development" with a new date recorded from the beginning of the development process. After the development process of the defect is finished, its status changes to "development completed" with respect to the recording date of finishing this process as bugs fixing is a time-consuming process (Hooimeijer, 2007). With small calculations of dates between "Under Development" and "Development Completed", we can measure how much time it took the development team to fix this defect. After the status "Development Completed" is finished, a Regression Test Phase is going to achieve. When finishing all test cases and scenarios for the defects, the quality control engineers release the status "Test Complete" then the status "Closed" for finishing the scenario.

The last scenario showed different statuses which defect moves through it, concerning the time element that recoded before and recording every change on defect status, as mentioned by Tammana and Faught (1998) "A defect tracking system that lets the user query the defect database is useful not only to generate summary reports, but also to track the status and the progress of a project that's underway" (Tammana, 1998). Therefore, through the power of DBMS (database management system), we can achieve a powerful tracking of defects through this last scenario.

The Following Table, Abbreviated the different status of tracking statues that the defects take through different phases of the product development phases and whose responsibility to check.

| | Status | Meaning | Responsibility |
|---|---|---|---|
| 1 | Initial | Declaration status for Fixing the Defect. | Quality Team |
| 2 | Under Development | Declaration status for starting Development on Defect. | Development Team |
| 3 | Development Complete | Declaration status for finishing development on defect | Development team leader |
| 4 | Under Test | Declaration status for starting test | Quality Team |
| 5 | Test Complete | Declaration status for finishing test | Quality Team Leader |
| 6 | User Acceptance Test Complete | Declaration status for user acceptance that the defect fixed | Users and Quality team leader |
| 7 | Closed | Declaration status for finishing all process needed for fixing defect | Project Manager |
| 8 | Need more Details | Declaration status for misunderstanding the requirement or function needed | Development Team |
| 9 | Postponed | Declaration status for postponing task for a time or next versions | Project Manager<br>Development Team Leader |

| | | | |
|---|---|---|---|
| 10 | Refused | Declaration status for Refusing the Task for unacceptable requirement or function needed. | Development Team leader |
| 11 | Reopen | Declaration status for reopening a defect with a same condition of a recorded defect before | Quality team leader |

**Table 1:** Proposed Defect status Scheme

## Conceptual Framework Design for the Proposed Defects Tracking Model

Based on the previous discussions in sections 1, 2 and that derived from the development of the research synthesis model for different ways of defect classifications that were presented in the preceding section, the ultimate conceptual model is given in Figure (1). The present research adopts the following model for classifying bugs which appear through different development phases especially in the maintenance and testing phases. As mentioned by Boehm and Basili, the maintenance phase consumes over 70% of the total life cycle cost of the software development projects (Boehm, 2001). The model developed was based on the previous work of (Edwards, 2006), (Janák, 2009) and on our general synthesized model for classifying and tracking defects (section 3 & 4). It is quite suitable for a case study. This model will guide us through our exploration for classification of defects to enhance quality control for the software development and maintenance processes.

Our model will focus, in details, on the phases of preventing defects and classifying them. Moreover, it concentrates on the bug examination, location and type factors for the insertion process of defect reports. Due to the highly exploratory nature of this research, all isolated conceptual variables/factors only represent the initial ideas about the discussion of defects tracking the phases and classification method to deal with in the future.

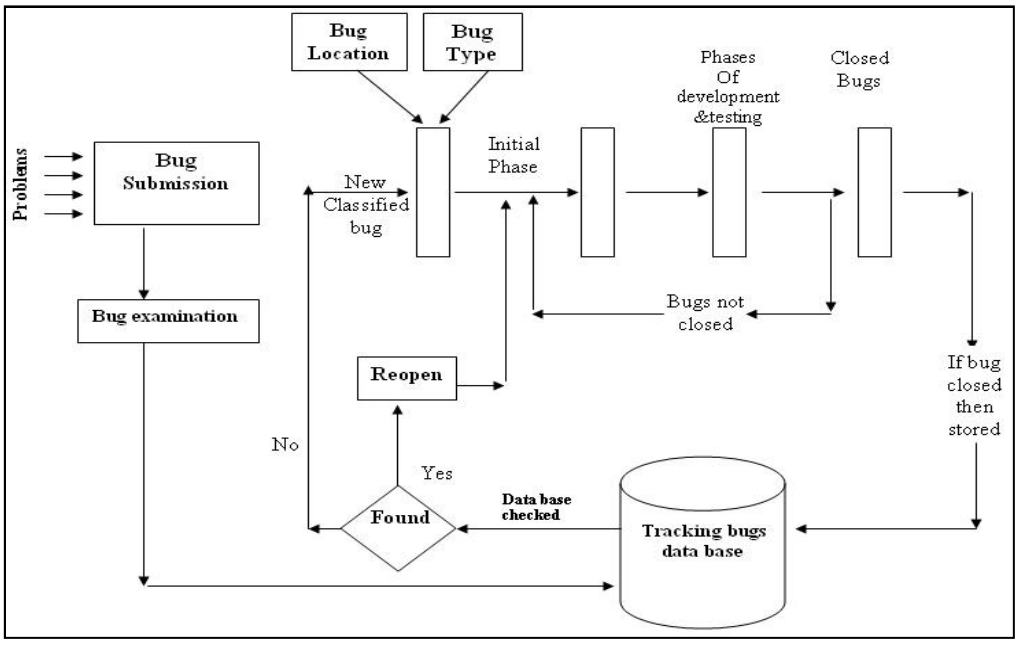


**Figure 1:** Proposed Defect Tracking Model

**Evaluation**

To evaluate the utility of the model, we performed a case study for locally made tracking tool used in an Egyptian software development company called (4s Systems). In addition, an experiment was made based on the data collected through the case study to achieve the purpose of evaluation. Also, we used in our research methodology observations, and formal interviews with a well prepared interviews protocol to face and analyze problems of defect tracking systems.

**Experiment Descriptions**

According to the conceptual framework of the proposed model, which built based on the discussed literature review, also the database gained and analyzed. We believe that experiments are suitable solutions for proving deductions. Different definitions of experiments and its advantages and disadvantages were cited in (Blaxter, 2006), (Kothari, 2004) and (SjØberg, 2005). However, according to the proposed model, we detected three factors (Bug Examination, Bug Location and Bug Type) which were different between this model and the others. An experiment was designed for measuring each factor separately and will be discussed in the next subsections.

**Bug Examination Factor**

We choose five random users from the quality control team that have at least two years experience. Also, we choose a random sample consisting of 50 defect reports for the overall data of all applications which were registered in the database. Afterwards, well defined and classified 20 defect reports were well prepared for the experiment. We divided the sample into two parts: First Part contains new ten bug scenarios and the Second Part had ten bug scenarios duplicated from the 50 bugs in the database. We requested the users to select randomly, four bug scenarios from the 20 bugs to register them with the tracking tools.

With re-doing the last designed experiment with new factors of bug examination that represents in proposed screen; new trends appeared. The function of the new examination screen was to force the user to check the historical defect database at least one per new record. Also, another function of the examination screen was to track the number for each user attempt to check the database before registering new record. This would help users in discovering the duplicated scenarios to be reopened (in reopen status). The results were Coordinated in the following Table 2.

**Bug Type Factor**

The experiment was designed using a random sample of defect reports that contain (50) bugs; then we classified these reports with the top managers of departments of the organization who have at least 20 years of experience.

The defect reports, which represents the data sample was classified according to the experience consulted top managers of the department and in parallel with the researcher's view to be a reference data after running the experiment.

All sample scenarios classified in a group with a serial number key to be easily detectable when participants were randomly taken. Also, we have discussed all bug types and what does it mean to all participants before beginning classifying and registering them on the dummy database as a step on the designed experiment.

**The Bug Location Factor**

The objective of the bug location experiment, was to classify the location of each bug scenario in a numeric path to become more precise in detecting and describing the bugs where they occur. Also, another objective was to measure the ability of the participants to use a unified method in classifying the paths of defect instead of being written thought the text. Hence, achieving the last objectives will help in restricting the inserted data about classifying the defects; as the user has no choices to register a new defect report except from classifying the path of defects.

According to the problem which faced the research in detecting the place where the bugs took place, we proposed to describe the places in a numeric manner, and in a classification structure as screen, report etc. However, the paths of all places through the application, were described in numeric path stored in relational database tables.

The experiment began with an empty database expected from the random sample consisting of (50) bugs that were previously classified. We designed a screen that would help the users who participate in the experiment, to classify the path of defects thought text of defects description scenario into controlled detectable choices. As the paths of bugs were described previously in the text without a unified method or detectable choices (e.g. file /choose registration screen / new record). Afterwards, we distributed the defect reports randomly and equally to the users after inserting the screen of application paths which was connected to the database. In addition, we requested them to detect the path of the bugs in the application after reading, and extracting them from the task description field. The path of the bugs usually described through the task description field in tracking tool used by the company.

**Experiment Results**

As mentioned in section 5.1 the model proposed three factors Bug Examination, Bug Location and Bug Type. The experiment results will be shown according to each factor separately in the next subsections.

**The Bug Examination Factor**

According to the results showed in the Table 2, we focused only on two dependent values i.e.: Registered and Reopen State. Also, we compare the two results after and before the experimentation. Analyzing the results specially the registered state after running the experimentation compared with the tracking users attempts factor, showed that there was a change of the curve values in figure 2.

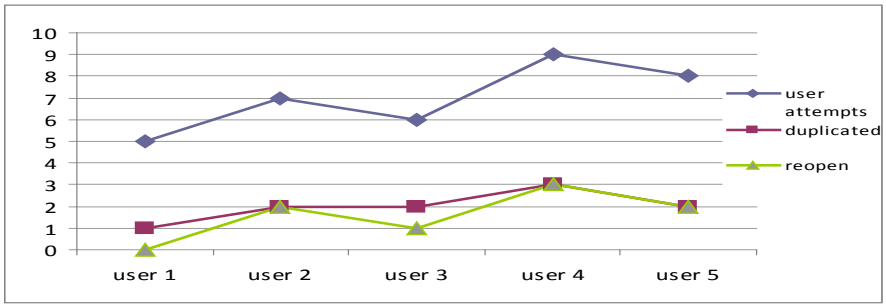


**Figure 2:** The Causal relationship between the Factors (User's attempts and reopen).

There was a positive relationship between the numbers of user's attempts to check for a registered defect with reopen factor corresponding to the duplicated factor of defects.

| | Tracking Users Attempts | Duplicated Bugs | New Bugs | Registered | Reopen State |
|---|---|---|---|---|---|
| User (1) | 5 | 1 | 3 | 4 | 0 |
| User (2) | 7 | 2 | 2 | 2 | 2 |
| User (3) | 6 | 2 | 2 | 3 | 1 |
| User (4) | 9 | 3 | 1 | 1 | 3 |
| User (5) | 8 | 2 | 2 | 2 | 2 |
| Total | | 10 | 10 | 12 | 8 |

**Table 2:** Results of examination factor after the experiment

**The Bug Type Factor**

| Bug type | Number of Standard Sample (x) | Total (T) | Means of Bug Type versus to Total Sample | After Registered (p) | Error Ratio (z) |
|---|---|---|---|---|---|
| Interface Error | 5 | 50 | 10% | 4 | 2% |
| Calculation Error | 12 | 50 | 24% | 10 | 4% |
| Loading Error | 10 | 50 | 20% | 10 | 0 |
| Security Error | 9 | 50 | 18% | 9 | 0 |
| Documentation Error | 2 | 50 | 4% | 2 | 0 |
| Enhancement Error | 4 | 50 | 8% | 5 | -2% |
| Business Logic Error | 8 | 50 | 16% | 10 | -4% |

**Table 3:** Results of Analysis of Bug Type Factor Data.

We used a well prepared standard sample data to each type of bug type factor which Referred to with factor (x). Also, we used simple statistical law Z= ((x-p)/T)*100 for measuring the error ratio where the factor (Z) is the error ratio between x and p versus the total sample.

The negative error ratio percentage (-2% & -4%) were interpreted to the incorrect defect reports registered as wrong in the defect types enhancement and business logic error attributed to human error factor.

**The Bug Location Factor**

| | Sample (x) | Total (T) | Mean | After Registered (p) | Error Ratio(z) |
|---|---|---|---|---|---|
| Screen | 20 | 50 | 40% | 20 | 0 |
| Report | 15 | 50 | 30% | 15 | 0 |
| Other | 15 | 50 | 30% | 15 | 0 |

**Table 4:** The Results of Bug Location Factor after the experiment

As shown in the last table, after using our proposed bug location factor of the proposed model with the standard sample data, we achieved a high percentage of success with zero error ratio percentage.

The results of error ratio with zero value confirmed the possibility of converting paths of errors that described in the reports of problems to be tracked digitally and easily recognizable by the human element used in the process of recording errors. Hence, we could uniformly unify the description process of different tracks within applications. This added a new element in the process of registering errors in addition to the previous factors the bug examination and bug type to facilitate future retrieval and tracking errors.

**Conclusions**

This Paper described terms and findings from significant earlier research, thereby forming a conceptual context and a foundation for the exploratory observational study that was central to this research.

The following are the Four Main Findings drawn from the study:

1-The Bug Examination Factor is considered as the greatest effect on the process of tracking defect reports. As with increasing the efforts of examining the tracking history of the database; it decreases the level of registered defect duplication.

2- The Bug type has significant effects on evaluating the tracking system as a method of structuring the inserted data to have structured correct information.

3- The Bug location has significant effects on detecting location of defects precisely through software development and enhancement.

4- The Quality Control field depending directly on human factor in triaging defects. This appears obvious through the Error Ratio in classifying defect type and bug location factor.

Considering the Limitations of the Conceptual Study Model, the following agenda for further research is proposed:

1-User's Authorization and Privileges through the submission phase of defect tracking models, have to be measured and enhanced.

2- Securing sensitive information for customers registered through the database tracking tools have to be researched.

**References:**

(Avram, 2007) Gabriela Avram, Anne Sheehan and Daniel K. Sullivan, Defect Tracking Systems in Global Software Development – a work practice study, Limerick, Ireland, 2007.

(Basili, 1984) Victor R. Basili, and Barry T. Perricone, Software Errors and Complexity: An Empirical Investigation, Communications of the ACM, P.42-52, 1984.

(Black, 1999) Rex Black, Managing the Testing Process, Wiley, 1999.

(Blaxter, 2006) Loraine Blaxter, Christina Hughes and Malcolm Tight , How to Research THIRD EDITION ,2006 , p135 .

(Boehm, 2001) Barry Boehm and Vector R. Basili, Software Defect Reduction Top 10 List, p.135–137, 2001.

(Curhan, 2005) Lisa A. Curhan, Software Defect Tracking during New Product Development of Computer System, Massachusetts institute of technology, 2005.

(Edwards, 2006) Jim Nindel-Edwards and Gerhard Steinke, A Full Life Cycle Defect Process Model That Supports Defect Tracking, Software Product Cycles, And Test Iterations, Communications of the IIMA ,Volume 6 Issue 1, 2006.

(Endres, 1975) Albert Endres, an Analysis of Errors and Their Causes in System Programs, Proceedings: International Conference on Reliable Software, P327-336, 1975.

(Fredericks, 1998) Michael Fredericks and Victor Basili , A State-of-the-Art-Report for Using Defect Tracking and Analysis to Improve Software Quality , 1998.

(Fry, 2010) Zachary P. Fry and Westley Weimer, A Human Study of Fault Localization Accuracy, 2010.

(Grady, 1996) Robert B. Grady, Software Failure Analysis for High-Return Process Improvement Decisions, Hewlett-Packard Journal, 47(4), 1996.

(Hooimeijer, 2007) Pieter Hooimeijer and Westley Weimer, Modeling Bug Report Quality, In Automated Software Engineering, pages 34–43, 2007.

(IBM, 2005) IBM® Rational® ClearQuest® Deployment Kit, Usage Model Guidelines, Version 1.1, Jan., p. 4, 2005.

(Jalbert, 2008) Nicholas Jalbert and Westley Weimer, Automated Duplicate Detection for Bug Tracking Systems, IEEE, 2008.

(Janák, 2009) Jiří Janák, Issue Tracking Systems, Brno, spring, 2009.

(Just, 2008) Sascha Just, Rahul Premraj and Thomas Zimmermann, Towards the Next Generation of Bug Tracking Systems, 2008.

(KO, 2003) Andrew J. KO and Brad A. Myers, Development and Evaluation of a Model of Programming Errors, 2003.

(Kothari, 2004) C.R. Kothari, research methodology methods and techniques (second revised edition), 2004, p121.

(Lenin, 2010) R.B. Lenin, R. B. Govindan and S. Ramaswamy, Predicting Bugs in Distributed Large Scale Software Systems Development, SCS M&S Magazine, 2010.

(Lethbridge, 2003) Timothy C. Lethbridge , Janice Singer and Andrew Forward , How Software Engineers Use Documentation: The State of the Practice, IEEE Software Volume 20 (Issue 6), 2003.

(Mays, 1990) R.G. Mays, C. L. Jones, G. J. Holloway and D. P. Studinski, Experiences with Defect Prevention, IBM Systems Journal, 29 (1) (January):4-32,1990.

(Microsoft, 2012) Process Guidance documentation provided in the documentation for Microsoft Visual Studio, 2012, http://msdn.microsoft.com/en-us/library/ee332488.aspx.

(Ostrand, 1984) Thomas S. Ostrand, and Elaine Weyuker, Collecting and Categorizing Software Error Data in an Industrial Environment, the Journal of Systems and Software, 4:289-300, 1984.

(Ostrand, 2004) Thomas J. Ostrand and Elaine J. Weyuker, A Tool for Mining Defect-Tracking Systems to Predict Fault-Prone Files , 1st International Workshop on Mining Software Repositories, united kingdom, 2004 .

(Pfleeger, 1998) Pfleeger, and Shari Lawrence, software engineering theory and practice upper saddel river, NJ: prentice hall, 1998.

(Rus, 2002) Ioana Rus, Combining Process Simulation and Orthogonal Defect Classification for Improving Software Dependability, Chillarege Press, 2002.

(SjØberg, 2005 ) D.I.K. SjØberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.-K.Liborg, A.C. Rekdal, A survey of controlled experiments in software engineering, IEEE Transactions on Software Engineering 31 (9),2005, P 733–753.

(Stimson, 1998) Carl Stimson, the Quality Improvement Story, 1998.

(Sullivan, 1992) Mark Sullivan and Ram Chillarege , A comparison of Software Defects in Database Management Systems and Operating Systems ,IEEE,1992 .

(Tammana, 1998) Prathibha Tammana and Danny Faught, Software Defect Isolation, 1998.

(Weinberg, 1983) Gerald. M. Weinberg, Kill That Code, Info systems, 1983, P.49.