INSTITUT FÜR SOFTWARETECHNOLOGIE

# Program File Bug Fix Effort Estimation Using Machine Learning Methods for Open Source Software Projects

**IST-TR-2009-04-24**

**Syed Nadeem Ahsan** **Javed Ferzund** **Franz Wotawa**

IST TECHNICAL REPORT

APRIL 2009

Institut für Softwaretechnologie, Inffeldgasse 16b/2, A-8010 Graz, Austria,
Phone +43-316-873-5711, Fax +41-316-873-5706
office@ist.tugraz.at, http://www.ist.tugraz.at/

IST TECHNICAL REPORT

IST TECHNICAL REPORT IST-TR-2009-04-24, APRIL 2009

# Program File Bug Fix Effort Estimation Using Machine Learning Methods

**Syed Nadeem Ahsan**[1] **Javed Ferzund**[2] **Franz Wotawa**[3]

**Abstract.** The bug fix effort estimation model for open source software system plays an important role in software quality assurance and software project management. Most of the effort estimation models are related to commercial or closed software systems, whereas it is difficult to develop an effort estimation model for open source software systems (OSS). Reasons may be the large number of source files, and the large number of software developers and contributors. In case of OSS, the developers and contributors are randomly distributed throughout the world. It is complicated to obtain the actual time spent by each developer or contributor to fix a bug. To overcome these issues we have applied a heuristic approach, which estimates the actual bug fix time. The total time spent to fix a bug is considered as bug fix effort. To perform experiments we selected the Mozilla open source project and extracted the bug fix activity data from the corresponding bugzilla server and downloaded source files revisions from CVS repository. We processed the program files bug fix change data and obtained a set of metrics, which are related to program file changes. To develop a reliable effort estimation model we used regression, neural network, support vector machine, classification rules and decision tree methods. Furthermore, we compared the outcome of the different methods using several evaluation criteria. The results show that the machine learning based models are better compared to the statistical regression model. In case of support vector machine the relative absolute error (MMRE) is lowest, while in case of classification rule the correlation between the actual and predicted effort values is highest.

[1] Institut für Softwaretechnologie, Technische Universität Graz, Inffeldgasse 16b/2, A-8010 Graz, Austria. E-mail: sahsan@ist.tugraz.at

[2] Institut für Softwaretechnologie, Technische Universität Graz, Inffeldgasse 16b/2, A-8010 Graz, Austria. E-mail: jferzund@ist.tugraz.at

[3] Institut für Softwaretechnologie, Technische Universität Graz, Inffeldgasse 16b/2, A-8010 Graz, Austria. E-mail: wotawa@ist.tugraz.at

# Program File Bug Fix Effort Estimation Using Machine Learning Methods for Open Source Software Projects

Syed Nadeem Ahsan, Javed Ferzund and Franz Wotawa
Institute for Software Technology
Technische Universität Graz
8010 Graz, Inffeldgasse 16b/2, Austria

{sahsan, jferzund, wotawa}@ist.tugraz.at

## ABSTRACT

The bug fix effort estimation model for open source software system plays an important role in software quality assurance and software project management. Most of the effort estimation models are related to commercial or closed software systems, whereas it is difficult to develop an effort estimation model for open source software systems (OSS). Reasons may be the large number of source files, and the large number of software developers and contributors. In case of OSS, the developers and contributors are randomly distributed throughout the world. It is complicated to obtain the actual time spent by each developer or contributor to fix a bug. To overcome these issues we have applied a heuristic approach, which estimates the actual bug fix time. The total time spent to fix a bug is considered as bug fix effort. To perform experiments we selected the Mozilla open source project and extracted the bug fix activity data from the corresponding bugzilla server and downloaded source files revisions from CVS repository. We processed the program files bug fix change data and obtained a set of metrics, which are related to program file changes. To develop a reliable effort estimation model we used regression, neural network, support vector machine, classification rules and decision tree methods. Furthermore, we compared the outcome of the different methods using several evaluation criteria. The results show that the machine learning based models are better compared to the statistical regression model. In case of support vector machine the relative absolute error (MMRE) is lowest, while in case of classification rule the correlation between the actual and predicted effort values is highest.

## Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *copyrights, cost estimation, life cycle, productivity, programming teams, software configuration management, software process models, software quality assurance, time estimation.*

## General Terms

Measurement, experimentation

## Keywords

Program files change, software repositories, software effort estimation, and process metrics.

## 1. INTRODUCTION

The timely delivery of a certain software product is an important requirement because it increases the market value of the product and allows for well-organized marketing activities if necessary. This holds especially for consumer products. Therefore having an accurate estimate of the schedule of the product release date is necessary for software product managers. Of course such estimation has to be based on estimates of the effort needed to develop the software products. In this paper we do not focus on providing an estimate from scratch. Instead we assume a software system where new releases are available on a regular base. This allows for an analysis of previous changes and the corresponding effort in order to estimate the effort of future changes.

During the evolution of a software system the source code lines of program files are frequently changed. The changes in source code are introduced either to enhance the product feature or sometime to fix faults that cause failure, which are detected when using the product. Because of available tools such as versioning systems like CVS and bug databases like bugzilla the whole evolution of a program is available for further analysis. Unfortunately the available data does not allow directly deriving estimation data, which holds especially in the open source software (OSS) domain. Moreover, the available databases are huge and can hardly be analyzed manually. Hence, an automated data extraction and analysis tool is necessary. The final objective is to have an estimator that can be automatically obtained from CVS and bugzilla databases and that estimates the effort necessary to implement certain changes.

Effort estimation in the OSS domain becomes more and more important because large multinationals software companies, such as IBM, Novell, and Microsoft also support OSS [7]. Therefore from the last few years more research work has been started in this domain. An accurate effort estimation model for OSS will further motivate the commercial organizations to continue their support for OSS. There are several other advantages of having such an estimator. First, the estimation of the effort to develop a software product provides an initial knowledge about the complexity of the product. Second, such an estimator allows for reliable and accurate scheduling of product release date. Accurate product release scheduling always increases the demand of the product and reduces the cost of the project. Moreover, an effort estimator allows for task assignment and resource management. Given all the mentioned advantages someone would expect a well-founded theory and several applications in the area of effort estimation from available data. Unfortunately it seems that the research community has ignored this key area in the field of software engineering. As a result we don't have enough theory and method for effort estimation [8].

Saying that there is not enough research in the domain of automated extraction of effort estimators we have to mention that there is a lot of research in the domain of estimation methods. Initially algorithmic estimation methods were used for software effort estimation, like COCOMO and COCOMO II and function point method [8]. These are based on historical data. However, these estimation techniques do not perform very well especially when they are used to estimate defect fix time [2]. The core reason may be that the defect fix time is not only based on counting of code lines or function points. Instead it is mainly based on the number of defects, the types of the defects, the expertise of the developer, the program file complexity, the number of changed lines of code (LOC) and number of deltas. Hence, all these aspects have to be taken into account when developing an automated method for extracting effort estimation from available data.

In this paper we present a method that uses the data from software repositories and bug databases in order to extract a predictor for estimating the effort necessary to fix a bug. There are many possibilities for predictor extraction, including linear regression and other data mining methods. In order to answer the question which method would give back the best, i.e., most accurate estimator, we tested several methods using the same data set. The data set comes from the OSS project Mozilla. In the paper we precisely show how the relevant data can be extracted and under which assumptions. In particular the real efforts spent for correcting a bug cannot be obtained from the databases. Hence, this information has to be estimated.

The paper is organized as follows: In Section 2 we discuss related work. In Section 3 we describe how we obtained the data from the repositories. Furthermore, we explain the used program file change metrics. In Section 4 we discuss about the effort estimation model and the empirical analysis of the results obtained. Finally, we conclude the paper and discuss future work.

## 2. RELATED WORK

There are several attempts on estimating efforts from available data reported in literature. H. Zeng et al. [13] used NASA's KC1 data set to present a solution for estimating software defect fix effort using self-organizing neural networks. Their experimental results are good when applied to estimate the effort for similar software development projects. Cathrin Weiss et al. [3] used the existing issue tracking system of the JBoss project. The basic idea behind their work is to identify the most similar earlier issued bug report and use the reported average time as an estimate. To predict the effort for new issue or bug report they used the nearest neighbor approach. Song et al. [12] used the NASA's SEL defect data set and applied association rule mining on the data set, to classify the effort using intervals. They found that association rule mining technique is better to predict the effort compared with other machine learning techniques like PART, C4.5 and Naïve Bayes. Lucas D. Panjer [4] used data mining tools to predict the time to fix a bug. The empirical evaluation is based on the Eclipse bug database. The proposed model can correctly predict up to 34.9% of the bugs into a discretized log scaled lifetime class.

Juan Jose Amor et al. [7] worked on free software and present an idea about how one can build an effort estimation model for OSS. The authors mentioned in their paper that effort is a function of the developers' activities that can be extracted from CVS, bug tracking systems, and the developers' emails. Although, the authors introduced a method for developing an effort estimation model they did not present such a model. Stefan Koch [11] discussed the issue of programmer participation and effort modeling for OSS. He worked on the GNOME project data and estimated the effort on the basis of programmer participation and the product metrics. He concluded that the effort estimation using programmer participation is lower than the estimation from product metrics.

Lucas Layman et al. [8] used the Microsoft project Visual Studio data for their research work. They collected the effort estimation data for 55 features of Visual Studio Team System and analyzed the data set for trend, patterns and differences. Their statistical analysis showed that the actual estimation error is positively correlated with the featured size and also the process metrics are correlated with the final estimation error.

Gary D. Boetticher [5] used neural network to develop an effort estimation model. He obtained effort data from a commercial company and also collected a set of product metrics like lines of code, Halsted vocabulary, object and complexity metrics. He designed multiple set of neural network based on different combination of hidden layers and the number of perceptrons per layer. He used different combination of product metrics to train the model. He performed the experiments using 33,000 different models of neural network, and collected the best data. His result shows that neural network can be used for an effort estimation model. But a huge number of experiments are necessary to obtain the best model. Most of the effort estimation work is related to commercial or close software system. Very few research papers for OSS are available, especially in the area of bug fix effort estimation. Most of the mentioned work use machine learning techniques to train the model using available effort data, whereas in our case we do not have effort data. Therefore, we have to extract the effort data from OSS to perform the experiments.

## 3. OBTAINING DATA FROM REPOSITORIES

In order to develop an effort estimation model we have to obtain the relevant data from the available repositories. These include the metrics data and an estimate of the effort needed to fix a certain bug. In particular we rely on the Mozilla CVS repository and Mozilla bug database. In case of another software project where a CVS repository or a bug database like bugzilla is available, all the information extraction described in this section can be directly applied. In other cases the described metrics and effort data may be extracted in a different way. However, the underlying concepts including which metrics to use and how to obtain the effort estimates can be re-used. The overall data extraction process for effort estimation is shown in Figure 1.

## 3.1 Metrics data

For obtaining the metrics we downloaded the selected revisions of the C++ program files from the Mozilla CVS repository and stored them on a local disk. We use an approach described in [1] to identify only the bug fix changes in a revision. In particular we classify the program file changes into four different types: bug fix, clean, bug introducing and bug fix-introducing changes. Note that we only consider the differences between two consecutive revisions, i.e., the release where the bug is fixed and its immediate predecessor, of each selected source file. These differences are processed to identify the type of program file changes. We store all the bug fix change data set into a database.

**Table 1. List of metrics**

| *Metrics* | *Description* |
|---|---|
| $S_{FCOUNT}$ | Source File Count: Number of source files which are changed to fix the bug. |
| $D_{COUNT}$ | Developer Count: Number of developers who are involved in fixing a bug. |
| $D_{EXP}$ | Developer Expertise: It is obtained by adding the rank values of all those developers who are involved in fixing a bug. |
| $T_{PREV}$ | Source File Age: It is obtained by adding all the previous revisions of the source files which are involved in fixing the bug. |
| $T_{PFREV}$ | Total previous fix revisions of source files. |
| $T_{LOC}$ | Total line of source code. |
| $T_{CLOC}$ | Total changed line of code. |
| $T_{DELTA}$ | Total number of change location in source files. |
| $T_{FINC}$ | Total number of included source files/packages. |
| $T_{FUNC}$ | Total number of function. |
| $T_{ELINE}$ | Total number of executable lines. |
| $T_{CYCLO}$ | Total cyclomatic complexity metrics. |
| $T_{PCOUNT}$ | Total number of parameter count. |
| $T_{RPOINT}$ | Total number of return points. |
| $T_{COPE}$ | Total number of changed operators. |

Table 1 shows a set of program file metrics, which we used to develop an effort estimation model. The list of metrics is not final; some other metrics may also be added in the list, like bug severity, bug voting etc.

To obtain the set of metrics we have processed the bug fix change data set and their related source files. We count the total number of revision, total number of bug fix revision of all the selected source files. The program change data are then processed to find out the total number of changed lines of code (LOC), total number of changed operators and the total number of delta (where delta is change hunk pair, we have at least one delta whenever a file is changed). We further obtain the number of developers and their expertise who were involved in fixing the bug from the bug report. A description of how to obtain and store the data we refer the reader to [1].

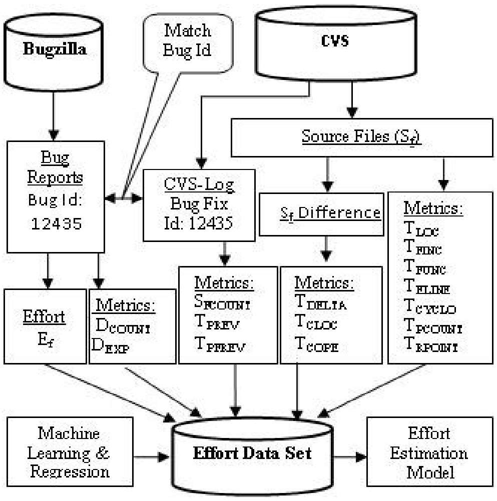
Bugzilla
Match Bug Id
CVS
Source Files ($S_f$)
Bug Reports Bug Id: 12435
CVS-Log Bug Fix Id: 12435
$S_f$ Difference
Metrics: $T_{LOC}$ $T_{FINC}$ $T_{FUNC}$ $T_{ELINE}$ $T_{CYCLO}$ $T_{PCOUNT}$ $T_{RPOINT}$
Effort $E_f$
Metrics: $D_{COUNT}$ $D_{EXP}$
Metrics: $S_{FCOUNT}$ $T_{PREV}$ $T_{PFREV}$
Metrics: $T_{DELTA}$ $T_{CLOC}$ $T_{COPE}$
Machine Learning & Regression
Effort Data Set
Effort Estimation Model

**Figure 1: Data extraction process**

For our further studies we downloaded 93,607 bug reports together with the bug fix history from the bugzilla repository. We processed the data and obtained the results depicted in Table 2. It can be seen that a large percentage of developers only fixes one bug. On the other side 4.9 percent of the developers are responsible for more than 70 percent of all bug reports. Hence, in the Mozilla project only a small percentage of developers can be considered as very experienced. This leads to our definition of developer expertise, where a developer has more expertise the more he or she is involved in fixing bugs. Or in other words, the developer expertise is directly proportional to the number of bug reports he or she has to handle. Table 2 shows the ranks of developers accordingly to their expertise in the Mozilla project. Only 212 developers can be assigned the highest rank according to their involvement in solving bug reports. Again a huge number of developers (2,442) are almost never involved in debugging activities.

## 3.2 Bug fix effort

In commercial software engineering different tools and techniques are available and used in companies to monitor and control the developer's monthly development activities. One of the techniques used is to maintain each developer's activity log. The developer's activity log provides the past and present development history of each developer. The log contains useful information including: the time spent to fulfill an assigned task and the time required to fix a certain bug.

In OSS development managing and monitoring the development activities is not that easy because most of the development task is performed by contributors which provide the solution in the form of patches. These patches are then validated and committed in the CVS server by the developers. In simple words developers are those who have rights to commit the changes in the source code whereas software contributors have no rights to commit the changes directly.

**Table 2. Number of bug reports assigned to developers**

| *No. of bug fixes Intervals* | *No. of developer for bug fixing* | *Expertise (Rank)* | *%age of developer for bug fixing* | *No. of bug reports fixed by developers* | *%age of bug reports assigned* |
|---|---|---|---|---|---|
| 00-01 | 2442 | 1 | 56.7 | 2442 | 2.7 |
| 02-03 | 624 | 2 | 14.5 | 1425 | 1.5 |
| 04-05 | 219 | 3 | 5.0 | 956 | 1.0 |
| 06-15 | 369 | 4 | 8.5 | 3476 | 3.7 |
| 16-25 | 148 | 5 | 3.4 | 2951 | 3.6 |
| 26-50 | 155 | 6 | 3.6 | 5797 | 6.2 |
| 51-100 | 131 | 7 | 3.0 | 9595 | 10.2 |
| >100 | 212 | 8 | 4.9 | 66965 | 71.5 |
| Total | 4300 | | | 93607 | |

Contributors are only allowed to submit possible solutions. Usually there are a large number of software contributors. Furthermore, they are spread over the world and they work usually independently. No specific tools are available to monitor and control the contributor development activities. Only bug reporting systems are used in OSS development to store the contributor's activities along with the developer's activities. The latter usually working as quality assurance. Another challenge is that the used version control systems only store the developers' information but not the contributors' ones. Hence, the necessary information of the effort required to solve a bug report has to be extracted from the bugzilla database and the CVS alone. In this section we discuss how to obtain an estimate of the effort.

Using bugzilla there is a dedicated bug life cycle. Figure 2 shows the bug life cycle. According to this life cycle, a bug starts as UNCONFIRMED. It immediately moves to the status NEW after it the bug is validated by the quality assurance (QA) person. Then it is moved to ASSIGNED, which is then followed by the RESOLVED status. Finally a bug may reach a status of VERIFIED, which is then followed by CLOSED. In some cases after the VERIFIED status a bug may goes to REOPEN again, and the cycle is repeated.

Let us now go into more detail of the bug life cycle. If a bug is said to be NEW, the QA person assigns the bug to any relevant developer/contributor in order to find a solution. Hence, the real work for fixing a bug starts when a bug is moved to the ASSIGNED state. The bug is solved when the developer or contributor provides a solution and moves the status to RESOLVED. The duration between ASSIGNED and RESOLVED is the actual period where effort is spend on source code changes to fix the bug. Hence, we assume that this time period is the actual bug fix time and thus the only time to be considered as effort. Note that in some cases a QA person reassigns the same bug to another developer or in some cases the previously assigned developer assigns the bug to some other developer, which makes the computation of the overall effort even more difficult. Since no information regarding the distribution of effort among the different developers is available, we assume that all developers contribute. Thus we calculate the sum of all time periods for each developer or contributor to come up with a single total bug fix effort.

For a single developer we compute the effort necessary to provide a solution to a bug report as follows: We start with the bug reports assigned to the developer. We compute the effort assigned to a bug report for each month of the year using the given dates for ASSIGNED and RESOLVED. The time span between ASSIGNED and RESOLVED cannot be used directly to compute the effort. The reason is that a developer works on several bug reports in parallel but it is impossible to spend more than all day's of a month in working. Hence, the time spend has to be multiplied by a factor. This factor takes into account the limited number of days available for working within a particular month

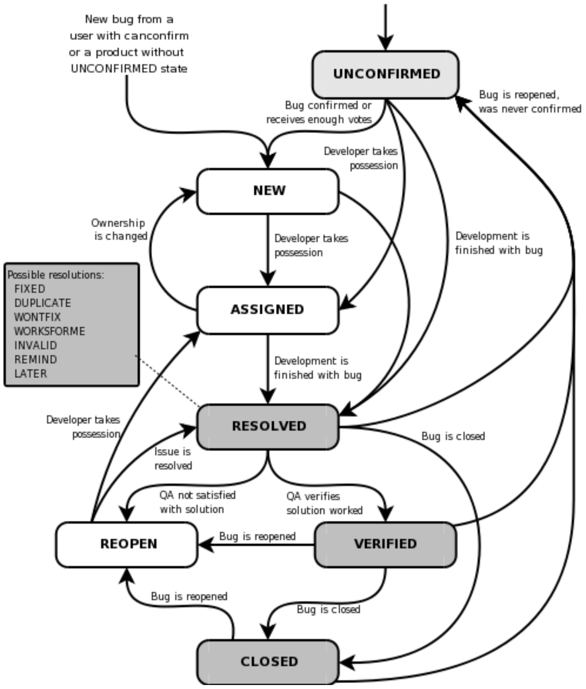
New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug is reopened, was never confirmed
Bug confirmed or receives enough votes
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions: FIXED DUPLICATE WONTFIX WORKSFORME INVALID REMIND LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Issue is resolved
Bug is closed
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

**Figure 2. The Bugzilla bug life cycle**

To understand how we have estimated bug fix effort from bug reports, consider an example in which a developer has fixed four bugs in the month of February. The example data is shown in Table 3. The green bar represents the durations in which the developer fixed those bugs. We obtained these durations from bug reports by subtracting the bug assigned date from the bug resolved date. In this example we assume that during some days of the month the developer worked in parallel on multiple bugs. Therefore we have to multiply

**Table 3. Bug fix effort estimation (in days) using bug reports**

<table>
<tr><td colspan="28">Developer Name: X Month: February</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="6">Bug Id= 1,<br>Bug fix duration 06 days</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td colspan="10">Bug Id= 2,<br>Bug fix duration 10 days</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td colspan="21">Bug Id= 3,<br>Bug fix duration 21 days</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="10">Bug Id= 4,<br>Bug fix duration 10 days</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td colspan="24">Total working days $\boldsymbol{T_w}$ = 24 days</td><td></td><td></td></tr>
</table>

each bug fix duration with a common multiplication factor ($M_k$). We obtain the multiplication factor by dividing the total actual working days of a month ($T_w$) with the sum of all the assigned working days for all the bugs

$$M_k = T_w /\sum Days$$
$$= 24/47 = 0.51$$
$$E_i = \sum (Bug\ fix\ duration\ for\ bug_i)_k \times M_k$$

where $E_i$ is the bug fix effort for bug $i$, and $k$ is the number of months spend to fix a bug. In this example $k$=1. Therefore, $E_1$ = 6×0.52 = 3.12. Similarly we can estimate effort for other bugs, $E_2$=5.1, $E_3$=10.7, & $E_4$=5.1

If a bug is fixed in more than one month, then the bug fix duration for the first month is obtained by subtracting the bug assigned day from the last day of the month, and for the last month, the bug fix duration is obtained by subtracting the first day of the month from the bug resolved day. We consider the whole days of a month as bug fix duration for all the intermediate months. We multiplied each month's bug fix duration with the respective multiplication factor. Finally, we add all the obtained values to get the effort value. Table 4 shows a typical assignment of bug reports to a developer obtained from the Mozilla bug database. Table 5 shows the results of the effort necessary to correct a bug.

The computation of effort spend to correct a bug described in a bug report assumes that the real effort is distributed evenly. This might not the case but since there is no other information available, it is the best we can do. This assumption as well as the others introduces some errors in the resulting data. But given the huge amount of data available in the repositories we expect that there is no error inherently built in this system of computing the effort. Hence, there might be a decrease of reliability in the data but there should always be an upper bound.

In the following sub-section we report on findings obtained using Mozilla project as an example.

## 3.3 Results obtained

All the data obtained are based on the bug fix activity data. We obtained these data by first extracting the bug information from the bug reports. We obtain these bug reports from the MSR 2007 data repository, which is freely available [3]. Then we obtained the developer activity data by extracting the information from the Mozilla bugzilla web server. In particular we downloaded all the activity data for all those bugs, which have been reported and fixed between Nov-99 and Dec-2007. We processed the downloaded HTML files and extracted the activity data set. Given this data we computed the effort and metrics as already described. For the latter we used the Mozilla CVS.

From the available data we are able to show some findings. These include a characteristic of bug reports and the bug effort over time, the distribution of bugs within the Mozilla project, and the distribution of the total number of bugs considering the days necessary to fix them. The latter is of particular interest because it gives an answer to the question for as much as a certain bug is more difficult to debugging than another.

Figure 3 shows a comparison between the total number of bugs fixed in a year and the corresponding average effort necessary to fix them. From this figure we conclude that during the first years of the Mozilla projects only small effort is required to fix a large number of bugs, whereas the number of bugs fixed per year decreases in the last years the average effort to fix them increases. This might be an indicator that the source code has reached a size and complexity where maintenance work is more and more difficult because a change in one file might influence others substantially.

Figure 4 depicts the bug fix effort for the Mozilla project for the different products comprising the software system. It can be seen that almost 50 % of the bug fix effort has been spent on bugs in the Core package of Mozilla. This is not really surprising because a bug in the Core package is most likely to be detected. Moreover, since the Core package implements the basic functionality changes as well as extensions might be implemented in the Core.

Figure 5 shows the relationship between the bug fix days and the frequency of bug occurrences again for the Mozilla project. The obtained relationship indicates that there are many bugs that can be corrected fast. Such a similar curve can be found in several different domains like in sales where only a few products are responsible for most of the turnover. However, and again similar to sale, there are bugs that are closed very lately (or product that are rarely purchased). Note that bugs requiring a lot of days for fixing need not to be the most complicated ones. It is more likely that they are not very relevant to run the Mozilla product most reliable. Otherwise, someone would try to fix the fault sooner.

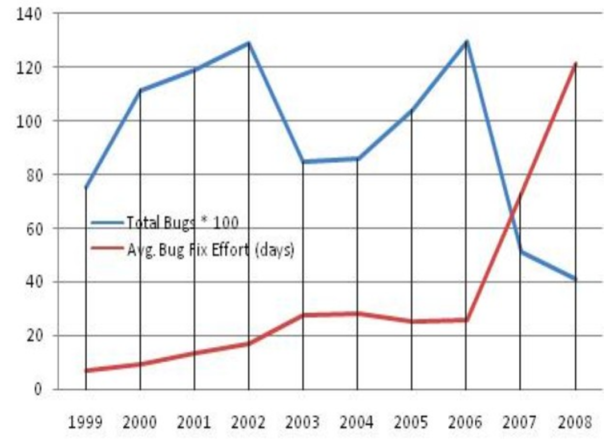

**Figure 3. The average bug fix effort and the number of bug fixes per year for the Mozilla project**

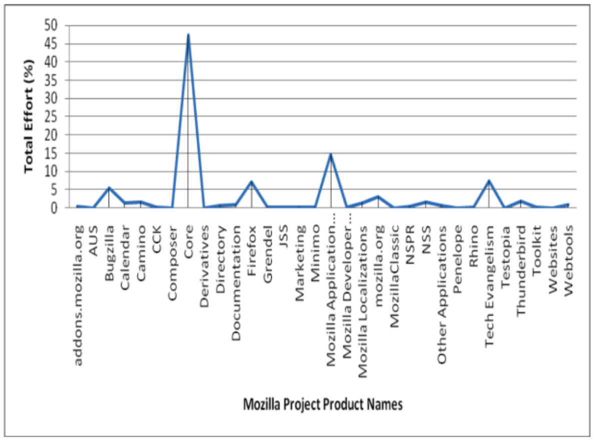

**Figure 4. The distribution of the relative effort spend to correct bugs for the different Mozilla product**

## 4. EFFORT ESTIMATION MODEL

The second and most important contribution of this paper is the comparison of different models for extracting an effort estimator from the obtained metrics and effort data. In this paper we compare the multiple linear regression model (MLR), and several machine learning (ML) methods, accordingly to their capabilities of serving as effort estimator. In the following we list the methods used in our experiments:

1. Multiple Linear Regression
2. Support Vector Machine (SVMreg),
3. Neural Network (Multilayer Perceptron),
4. Classification Rule (M5Rules) and
5. Decision Tree (REPTree, M5P).

MLR is a statistical method commonly used for numerical prediction. We used MLR because it is a simple tool and most commonly used for effort estimation purpose [10]. Support vector machines (SVM) belong to the class of supervised learning methods, which are used for classification and regression. SVM algorithms are also used for numerical prediction. One example is SVMreg. They generate a model that can be expressed in terms of a few support vectors and may be used in non-linear problem solving using the kernel function. Neural networks are commonly used for numerical prediction and effort estimation. We used multilayer perceptrons, which is belong to the *feedforward* class of networks. In *feedforward* type of network the output only depends on the current input instances and there are no cycles. M5Rules generates a decision list for regression problems. It builds a model tree using M5 and makes the best leaf into a rule. Decision trees classify input instances by sorting them according to feature values. Each node in a decision tree represents a feature, and each branch represents a feature value. REPTree is a decision tree based machine learning algorithm. It is a fast tree learner that uses reduced error pruning. M5P is the model tree learner. Model tree combines decision tree with the possibility of linear regression function at the leaves [6].

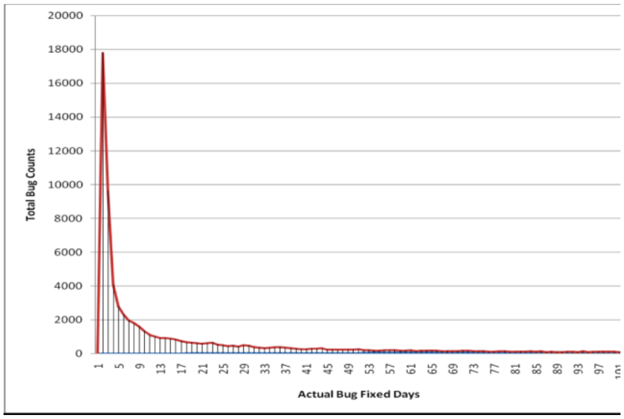

**Figure 5. Number of bugs versus bug fix days**

For the comparison we use the data obtained from the Mozilla project repositories. Moreover, the comparison is based on the following error measures. The mean absolute error (MAE), root mean square error (RMSE), the mean relative absolute error (MRE) and the mean magnitude of relative error (MMRE), the root relative square error (RRSE), the Pearson correlation coefficient R, and the percentage of prediction PRED(x). According to Conte et al [10], the MMRE value for effort prediction model should be ≤ 0.25. PRED(x) is obtained from relative error. Mostly PRED(0.25) is used but PRED(0.30) is also acceptable. To be self-contained we give the definitions of the error measures except the correlation coefficient. For the latter we refer to standard textbooks on statistics. Obviously a correlation value close to 1 means that the prediction and the real data fit well and the model is a good predictor. A value less than 0.5 indicate a poor performance of the prediction model. In the following definitions $E_{ACT}$ stands for the actual effort, $E_{PRED}$ for the predicted effort, $n$ for the total number of observed values, and $m$ for the number of correctly predicted values.

**Table 4. Bug assigned to developer neil@httl.net in 2001**

| Date Assigned | Date Resolved | Developer Name: neil@httl.net |  |  |  |  |  |  | Log Year 2001 |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | Jan | Feb | Mar | Apr | May | Jun | July | Aug | Sep | Oct | Nov | Dec |
| 2001-04-25 | 2003-07-31 |  |  |  | Bug Id: 1697 |  |  |  |  |  |  |  |  |
| 2000-05-16 | 2003-04-07 | Bug Id: 38367 |  |  |  |  |  |  |  |  |  |  |  |
| 2000-12-28 | 2001-02-09 | Bug Id: 54175 |  |  |  |  |  |  |  |  |  |  |  |
| 2001-12-11 | 2005-03-02 |  |  |  |  |  |  |  |  |  |  |  | 58523 |
| 2001-10-12 | 2004-11-28 |  |  |  |  |  |  |  |  |  |  | Bug Id: 58523 |  |
| 2001-03-27 | 2001-08-15 |  |  | Bug Id: 66475 |  |  |  |  |  |  |  |  |  |
| 2001-07-27 | 2003-02-28 |  |  |  |  |  |  | Bug Id: 75686 |  |  |  |  |  |
| 2001-05-14 | 2003-03-20 |  |  |  |  | Bug Id: 80837 |  |  |  |  |  |  |  |
| 2001-06-15 | 2003-09-11 |  |  |  |  |  | Bug Id: 85908 |  |  |  |  |  |  |
| 2001-06-26 | 2003-09-11 |  |  |  |  |  | Bug Id: 87924 |  |  |  |  |  |  |
| 2001-07-04 | 2003-10-17 |  |  |  |  |  |  | Bug Id: 89212 |  |  |  |  |  |
| 2001-08-29 | 2005-09-30 |  |  |  |  |  |  |  | Bug Id: 97532 |  |  |  |  |
| 2001-10-08 | 2002-12-12 |  |  |  |  |  |  |  |  |  | Bug Id: 99328 |  |  |
| 2001-11-16 | 2003-01-17 |  |  |  |  |  |  |  |  |  |  | Bug Id: 107418 |  |
| 2001-12-02 | 2002-01-21 |  |  |  |  |  |  |  |  |  |  |  | 110254 |
| 2001-11-27 | 2001-11-28 |  |  |  |  |  |  |  |  |  |  | 111606 |  |
| 2001-12-11 | 2002-01-15 |  |  |  |  |  |  |  |  |  |  |  | 114522 |
| 2001-12-20 | 2002-09-13 |  |  |  |  |  |  |  |  |  |  |  | 116196 |

**Table 5. The obtained bug fix activity log of developer neil@httl.net**

| Developer Name: neil@httl.net |  |  |  |  |  |  |  |  |  | Log Year: 2001 |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bug Id | Date Assigned | Date Resolved | Jan | Feb | Mar | Apr | May | Jun | July | Aug | Sep | Oct | Nov | Dec | Total Days (Effort) |
| 1697 | 2001-04-25 | 2003-07-31 | 00.00 | 00.00 | 00.00 | 02.31 | 08.74 | 06.47 | 04.43 | 04.11 | 03.75 | 03.31 | 02.86 | 02.28 | 038.26 |
| 38367 | 2000-05-16 | 2003-04-07 | 15.50 | 21.19 | 27.46 | 13.85 | 08.74 | 06.47 | 04.43 | 04.11 | 03.75 | 03.31 | 02.86 | 02.28 | 113.95 |
| 54175 | 2000-12-28 | 2001-02-09 | 15.50 | 06.81 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 022.31 |
| 58523 | 2001-12-11 | 2005-03-02 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 001.47 |
| 66475 | 2001-10-12 | 2004-11-28 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 02.03 | 02.86 | 007.17 |
| 72481 | 2001-03-27 | 2001-08-15 | 00.00 | 00.00 | 03.54 | 13.85 | 08.74 | 06.47 | 04.43 | 01.99 | 00.00 | 00.00 | 00.00 | 00.00 | 039.02 |
| 75686 | 2001-07-27 | 2003-02-28 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.57 | 04.11 | 03.75 | 03.31 | 02.86 | 02.28 | 016.88 |
| 80837 | 2001-05-14 | 2003-03-20 | 00.00 | 00.00 | 00.00 | 00.00 | 04.79 | 06.47 | 04.43 | 04.11 | 03.75 | 03.31 | 02.86 | 02.28 | 032.00 |
| 85908 | 2001-06-15 | 2003-09-11 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 03.24 | 04.43 | 04.11 | 03.75 | 03.31 | 02.86 | 02.28 | 023.98 |
| 87924 | 2001-06-26 | 2003-09-11 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.86 | 04.43 | 04.11 | 03.75 | 03.31 | 02.86 | 02.28 | 021.60 |
| 89212 | 2001-07-04 | 2003-10-17 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 03.86 | 04.11 | 03.75 | 03.31 | 02.86 | 02.28 | 020.17 |
| 97532 | 2001-08-29 | 2005-09-30 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 02.60 | 03.75 | 03.31 | 02.86 | 02.28 | 012.46 |
| 99328 | 2001-10-08 | 2002-12-12 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 02.46 | 02.86 | 02.28 | 007.60 |
| 107418 | 2001-11-16 | 2003-01-17 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 01.33 | 02.28 | 003.61 |
| 110254 | 2001-12-02 | 2002-01-21 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 02.14 | 002.14 |
| 111606 | 2001-11-27 | 2001-11-28 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.10 | 00.00 | 000.10 |
| 114522 | 2001-12-11 | 2002-01-15 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 01.47 | 001.47 |
| 116196 | 2001-12-20 | 2002-09-13 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.00 | 00.81 | 000.81 |
| Total no. days used to fix a bug |  |  | 31.00 | 28.00 | 31.00 | 30.01 | 31.01 | 29.98 | 26.58 | 31.00 | 30.00 | 28.94 | 29.20 | 30.08 | 365 |
| Number of bugs fixes per month |  |  | 2 | 2 | 2 | 3 | 4 | 6 | 8 | 9 | 8 | 11 | 12 | 14 |  |

$$MAE = \frac{\sum \left| E_{ACT} - E_{PRED} \right|}{n}$$

$$RMSE = \sqrt{\frac{\sum \left( E_{ACT} - E_{PRED} \right)^2}{n}}$$

$$MRE = \frac{\left| E_{ACT} - E_{PRED} \right|}{E_{ACT}}$$

$$MMRE = \frac{\sum_j MRE_j}{n}$$

$$RRSE = \sqrt{\frac{\sum \left( E_{ACT} - E_{PRED} \right)^2}{E_{ACT}}}$$

$$PRED(x) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1 & if \ MRE_i \le x \\ 0 & otherwise \end{cases}$$

To perform the experiment we used freely available Java based machine learning tool known as WEKA [6]. WEKA contains a large number of Java libraries of machine learning algorithms.

## 4.1 Multiple Linear Regression Model

To develop an effort estimation model using statistical MLR, we used a set metrics as a set of estimator for the model, and effort as a dependent variable. The set of metrics is shown in Table 6. Beside the used metrics Table 6 also comprises the correlation of the metrics with the effort, i.e., the number of days required to fix a bug. We further use a statistical tool R for obtaining a multiple linear regression model for effort estimation. The tool returns the following multiple linear regression model as output

$$EFFORT=-13.8\times D_{COUNT} -1.09\times D_{EXP}+5.8\times S_{FCOUNT}- 0.01\times T_{PREV} +0.02\times T_{PFREV} + 0.001\times T_{LOC}+ 0.021\times T_{CLOC}-0.035\times T_{DELTA}-0.034\times T_{FUNC}+ 0.004\times T_{ELINE} - 0.025\times T_{CYCLO} + 0.02\times T_{PCOUNT} -12.9$$

For evaluation purposes we used this model on a dataset, which comprises 7027 records. Each record consists of a set of metrics which is related to a single unique bug fix transaction of the source file revisions. The obtained result is shown in Table 7. The Pearsons correlation coefficient value for the correlation between the actual and the predicted effort value is 0.53. This indicates that the predicted model is correlated with the actual effort values.

To further analyze the effort model we compute the MMRE for the multiple linear regression model. The MMRE value is 84% that is not according to the standard acceptable value of MMRE, which is. 25-30%. The results of model accuracy estimation are shown in Table 7.

In the following we discuss possible reasons, why the correlation and MMRE value of our model is not matching the standard MMRE value for effort models. One reason might be some outliers in the data set. It is very common observation that predictors based on regression model are highly infected by the outliers of a data source. The other issue is MLR forced linear relationship among the data point. Figure 6 show the actual and predicted values of the bug fix effort using our obtained model. The trend of the graph shows that the actual and the predicted values have the same increasing and decreasing patterns.

**Table 6. Correlation of metrics with bug fix days (effort) for the Mozilla project**

| Sr. | Metrics | Pearson Correlation | Mean | Standard Deviation |
|---|---|---|---|---|
| 1 | $S_{FCOUNT}$ | 0.32 | 1.78 | 1.625 |
| 2 | $D_{COUNT}$ | 0.41 | 2.18 | 1.545 |
| 3 | $D_{EXP}$ | 0.36 | 15.17 | 10.082 |
| 4 | $T_{PREV}$ | 0.19 | 224.39 | 289.367 |
| 5 | $T_{PFREV}$ | 0.18 | 144.15 | 199.513 |
| 6 | $T_{LOC}$ | 0.20 | 2471.9 | 2799.26 |
| 7 | $T_{CLOC}$ | 0.18 | 39.43 | 85.450 |
| 8 | $T_{DELTA}$ | 0.22 | 9.97 | 20.829 |
| 9 | $T_{FINC}$ | 0.21 | 52.78 | 59.658 |
| 10 | $T_{FUNC}$ | 0.21 | 103.33 | 115.332 |
| 11 | $T_{ELINE}$ | 0.19 | 2200.8 | 2425.90 |
| 12 | $T_{CYCLO}$ | 0.20 | 424.04 | 482.832 |
| 13 | $T_{PCOUNT}$ | 0.21 | 179.42 | 200.111 |
| 14 | $T_{RPOINT}$ | 0.18 | 171.35 | 215.412 |
| 15 | $T_{COPE}$ | 0.13 | 29.28 | 67.397 |

## 4.2 Machine Learning Model

In order to develop an effort estimation model using machine-learning algorithms, we imported the data into WEKA and used 10 fold cross validation techniques for each ML method, which actually divide the data set into 10 equal parts and randomly select 9 parts for training and 1 part for testing and repeat it for 10 times.

For support vector machine SVMreg, we used the filter data type i.e. normalized training data. In case of multi layer perceptron, we designed multiple neural networks using 1, 2, 3 and 4 hidden layers with 2, 4-2 and 6-4-2 perceptron per hidden layer, and set the number of learning steps to value between 500-1000. For M5Rules method we used two classification rules on the basis of the $D_{COUNT}$ metrics value. We used the fast decision tree learner method REPTree with the pruning option. Table 7 depicts the obtained results of the learned models when using our Mozilla data set. We see that each model has a good correlation value. The same holds for the absolute error values but not for the relative absolute error values, which is not according to the recommended value i.e. between 25% and 30%. According to Table 7 the highest correlation value is for the classification rule M5Rules i.e., 0.56 and its MMRE value is 74%. In software effort estimation

**Table 7: Results obtained for machine-learning based effort predictor model using the Mozilla data set**

| Sr. No | Method Name | R | MAE | RMSE | MMRE (%) | RMRE (%) |
|---|---|---|---|---|---|---|
| 1 | Multiple Linear Regression | 0.53 | 11.88 | 23.5 | 81.0 | 84.7 |
| 2 | Support Vector Regression (SVMreg) | 0.51 | 9.36 | 26.7 | 63.8 | 95.8 |
| 3 | Neural Network (Multilayer Perceptron) | 0.54 | 29.2 | 58.2 | 93.6 | 86.3 |
| 4 | Classification Rule (M5Rules) | 0.56 | 10.8 | 23.0 | 73.6 | 82.5 |
| 5 | Decision Tree (REPTree) | 0.51 | 10.8 | 23.9 | 74.23 | 86.0 |
| | Decision Tree (M5P) | 0.55 | 10.6 | 22.9 | 77.7 | 82.5 |

modeling field, it is a common practice that the best effort model is the one that has the lowest relative absolute error value and high correlation value between actual and predicted effort values. Therefore in our experiment the best model is the classification rule method based on M5Rules. There are several reasons for why MMRE value is high. One reason may be the presence of noise or outlier in the data set. The other may be the quality of the effort data set because we don't get it from any organization; rather we extract it by our own heuristic method.

The result can be improved by doing the following steps

I. Remove outliers from the data set manually or apply clustering method.
II. Further improve the quality of the effort data.
III. Add some other metric into model, which should have high correlation with effort data.

## 5. CONCLUSION AND FUTURE WORK

In this paper we discuss the use of program file metrics and developer bug fix activity log information for estimating the effort required to fix a bug in the domain of open source software development. Since some data is not available like the effort spent for fixing previously corrected faults, we develop a method for deriving this information from bug repositories. We processed the resulting developers log data and obtained the bug fix effort values in terms of bug fix days. Besides giving some empirical data related to the Mozilla project, we focus on the development of effort models. In particular, we are interested in how well established techniques for knowledge extraction from data can be used for this purpose.

In this paper we give results obtained from statistical methods as well as from machine-learning approaches both based on the same underlying metrics and effort data. We trained the model by using the program file metrics and program file change metrics data of C++ source files from the Mozilla project repository. The correlation value between actual and predicted effort is 56% for Classification Rule M5Rules based model, which is a good result. Table 7 shows that all the machine learning and multiple linear regression models have correlation values between 0.51 and 0.56, and similarly the MMRE values lie between 63 and 93. This shows that the performance of our models is good, but can be further improved by removing the noise from the data set. In order to improve the quality of the data we plan to use the clustering method or some other method to remove the outliers from the data set. We will also work on finding other metrics, which have good correlation with the bug fix effort.

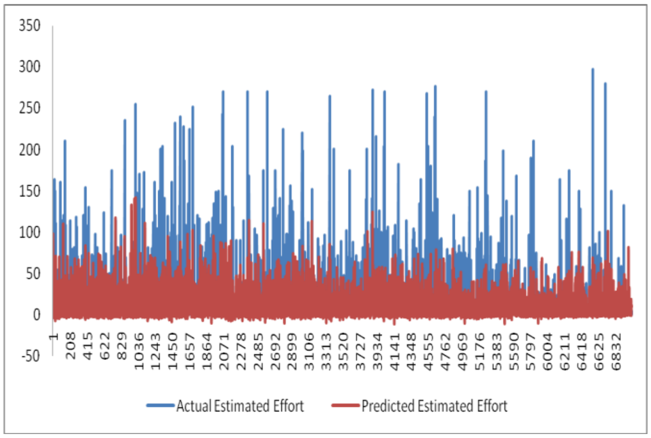

**Figure 6: Actual versus predicted effort for bug fixing using multiple linear regression model**

## ACKNOWLEDGEMENTS

The research work presented in this paper is partially funded by Higher Education Commission (HEC), Pakistan under HEC overseas scholarship programme and partially conducted within the competence network Softnet Austria (www.soft-net.at) that is funded by Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Witschaftsfrderum gsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and Technology (ZIT).

## REFERENCES

[1] Syed Nadeem Ahsan, Javed Ferzund, Franz Wotawa, A Database for the Analysis of Program Change Patterns, *Proceedings of the Fourth International Conference on Networked Computing and Advanced Information Management*. pp.32-39, 2008

[2] Boehm, B., et al., Cost Models for Future Software Life Cycle Process: COCOMO 2, Annals of Software Engineering, 1995.

[3] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. 2007. How Long Will It Take to Fix This Bug? In *Proceedings of the Fourth international Workshop on Mining Software Repositories* (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 1. DOI= http://dx.doi.org/10.1109/MSR.2007.13

[4] Lucas D. Panjer. Predicting Eclipse Bug Lifetimes. In *Proceedings of Mining Software Repositories (MSR-07)*. May 2007 Page(s):29 – 29.

[5] Boetticher, G., An Assessment of Metric Contribution in the Construction of a Neural Network-Based Effort Estimator, *Second International Workshop on Soft Computing Applied to Software Engineering*, 2001.

[6] Ian H. Witten. Data Mining Practical Machine Learning Tools and Technique. Second Edition, 2005. (Morgan Kaufmann Series in Data Management Systems)

[7] Amor, J. J., Robles, G., and Gonzalez-Barahona, J. M. 2006. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 international Workshop on Economics Driven Software Engineering Research* (Shanghai, China, May 27 - 27, 2006). EDSER '06. ACM, New York, NY,3-6.

[8] Layman, L., Nagappan, N., Guckenheimer, S., Beehler, J., and Begel, A. 2008. Mining software effort data: preliminary analysis of visual studio team system data. In *Proceedings of the 2008 international Working Conference on Mining Software Repositories* (Leipzig, Germany, May 10 - 11, 2008). MSR '08. ACM, New York, NY, 43-46. DOI= http://doi.acm.org/10.1145/1370750.1370762

[9] Kim, S. and Whitehead, E. J. 2006. How long did it take to fix bugs? In *Proceedings of the 2006 international Workshop on Mining Software Repositories* (Shanghai, China, May 22 - 23, 2006). MSR '06. ACM, New York, NY, 173-174. DOI= http://doi.acm.org/10.1145/1137983.1138027

[10] Conte, S. D., Dunsmore, H. E., and Shen, Y. E. 1986 *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc.

[11] Koch, S., Effort Modeling and Programmer Participation in Open Source Software Projects, *Information Economics and Policy*, 20(4): 345-355. 2008

[12] Qinbao Song, Martin Shepperd, Michelle Cartwright, Carolyn Mair, Software Defect Association Mining and Defect Correction Effort Prediction, *IEEE Transactions on Software Engineering*, vol. 32, no. 2, pp. 69-82, Feb., 2006

[13] Hui Zeng , David Rine, Estimation of Software Defects Fix Effort Using Neural Networks, Proceedings of the 28th Annual International Computer *Software and Applications Conference - Workshops and Fast Abstracts - (COMPSAC'04)*, p.20-21, September 28-30, 2004