Evaluating Predictive Models of Software Quality

V Ciaschini, M Canaparo, E Ronchieri, D Salomoni
INFN CNAF, Bologna 40126, Italy
E-mail: elisabetta.ronchieri@cnaf.infn.it

Abstract. Applications from High Energy Physics scientific community are constantly growing and implemented by a large number of developers. This implies a strong churn on the code and an associated risk of faults, which is unavoidable as long as the software undergoes active evolution. However, the necessities of production systems run counter to this. Stability and predictability are of paramount importance; in addition, a short turn-around time for the defect discovery-correction-deployment cycle is required. A way to reconcile these opposite foci is to use a software quality model to obtain an approximation of the risk before releasing a program to only deliver software with a risk lower than an agreed threshold.

In this article we evaluated two quality predictive models to identify the operational risk and the quality of some software products. We applied these models to the development history of several EMI packages with intent to discover the risk factor of each product and compare it with its real history. We attempted to determine if the models reasonably maps reality for the applications under evaluation, and finally we concluded suggesting directions for further studies.

1. Introduction

Although the quality-concept of software changes depending on user perspectives as stated by various authors [1], [2], a set of attributes and metrics exist to formalize and measure software characteristics according to defined unambiguous rules [3]. In this area researchers concentrate efforts to predict whether products and processes will meet goals for quality during the development life cycle. Many papers support models that profess to minimize the number of defects in software by predicting where they are likely to be detected [4]. The main context of this kind of models is given by the metrics [5] released by NASA regarding its own software projects [6]; however, open source systems are also considered. Over half of the studies of predictive models have analyzed code written in C or C++, and 20% of them are for code in Java. So far there are several challenges to be tackled: the identification of the right predictive model; the addressing of the known software capabilities, such as adaptability, maintainability, and reliability; the approximation of the risk factor for having faulty software packages before releasing a program; and the estimation of the number of defects.

The European Middleware Initiative (EMI) [7] was a project that aimed to develop and maintain software products of several partners in the context of the Large Hadron Collider experiment at CERN. In this study, we evaluated the operational risk and software quality of some EMI products under the INFN responsibility and still under development by using two quality predictive models with the discriminant analysis and regression method. We attempted to determine if the models reasonably map reality of the selected software products. The considered inputs of these models are divided in dependent and independent variables: the former are parameters derived from the size of the code, for example the number of lines of
code, the number of classes defined per file, and the number of components defined per software project; the latter are static code metrics (the most common basis for this kind of models), and previous fault data that express a continuing faulty behaviour of a piece of code. The outputs are expressed by dependent variables that we chose to highlight fault-prone software products. At the end of this study, we learnt that these models offer a great chance to improve the software development process. A previous review of the discriminant analysis model showed a high mathematical precision in estimating the risk factor. However, an increasing amount of data can improve the precision rate. We aim, therefore, to reach the same accuracy with the regression model by even introducing a new function that considers the measured data. Despite the initial effort to learn and understand predictive models, they may suggest which software products are more error prone and should receive greater attention (during the testing and quality assurance phases). The paper is organized as follows. In Section 2, a detailed description of the study is presented. Section 3 provides a first analysis on the raw collected data, while Section 4 provides a second analysis on the statistical data. Finally, Section 5 gives some conclusions on the performed evaluation.

2. Study Description
The experiment consisted of evaluating software quality of some EMI products in all the EMI distributions [8] by using predictive models.

As first approximation, we selected the INFN software products from the EMI project to analyze code coming from different development environments of the same institute but belonging to the same application fields (that are mainly related to the High Energy Physics community): CREAM (Computing Resource Execution And Management), a simple, lightweight service for job management operation at the computing element level [9]; StoRM (STOrage Resource Manager), an implementation of the standard SRM interface for disk-based storage solutions [10]; VOMS (Virtual Organization Management System), an attribute authority service [11]; WMS (Workload Management System), an implementation of the early binding approach to meta-scheduling [12]; WNoDeS (Worker Nodes on Demand Service), a solution to virtualize computing resources and to make them available through local, Grid and Cloud interfaces [13]; and parts of YAIM (Yet Another Installation Manager), a well-known and widely used grid services configuration tool [14]. As a consequence, we considered their software packages (see table 1) released in correspondence of the three EMI distributions - EMI 1 Kebnekaise, EMI 2 Matterhorn, and EMI 3 Monte Bianco. WNoDeS differs from the other software products since it was introduced starting from EMI 2. The source types are heterogeneous because of their use of different programming languages.

To address this study we decided to choose the most significant static metrics, which are part of the product category that provides results of the software development activities [15]. This category contains metrics that for example characterize the product by size, complexity and quality. They are directly or indirectly measurable by considering their attributes: those internal, such as Lines Of Code (LOC), have direct measures, while those external such as quality and complexity have indirect measures. The tabbed metrics are subdivided in size, quality and complexity types. The subset of the size metrics consist of: the Number Of Files (NOF), and the Number Of Extensions (NOE) in the software package; the Blank Lines Of Code (BLOC), the Comment Lines Of Code (CLOC), and LOC found in the files of the software package; and the Number Of Programming Languages (NOPL) supported in the software package. They may contribute to notice that a certain type or method might be hard to maintain. The considered complexity metric is the McCabe cyclomatic [16] that determines the complexity of a section of source code by measuring the number of linearly independent paths in the flow of the source code. It is based on a graph flow representation of the program, where nodes express processing tasks and edges show the control flow between nodes. An interpretation of the McCabe metric
is that a complex control flow will require more tests to achieve good code and will penalize its maintainability. The chosen quality metric is the number of reported defects of each release, which we denote Number of Defects (NOD).

Table 1. Software packages for the specified software product in all the EMI distributions: x means that the specified source package in the given EMI distribution has been updated.

| Products | EMI 1 | EMI 2 | EMI 3 |
|----------|-------|-------|-------|
| CREAM    |       |       |       |
| glite-ce-cream-1.13.x-x | glite-ce-cream-1.14.x-x | glite-ce-cream-1.13.x-x |
| glite-ce-cream-api-java-1.13.x-x | glite-ce-cream-api-java-1.14.x-x | glite-ce-cream-api-java-1.13.x-x |
| glite-ce-cream-client-api-1.13.x-x | glite-ce-cream-client-api-1.14.x-x | glite-ce-cream-client-api-1.13.x-x |
| glite-ce-cream-utils-1.13.0-3 | glite-ce-cream-utils-1.14.0-3 | glite-ce-cream-utils-1.13.0-3 |
| glite-ce-yaim-cream-ce-4.2.x-x | glite-ce-yaim-cream-ce-4.3.x-x | glite-ce-yaim-cream-ce-4.4.x-x |
| StoRM    |       |       |       |
| storm-backend-server-1.x.x.x | storm-backend-server-1.x.x.x | storm-backend-server-1.11.0-43 |
| storm-common-1.1.x.x | storm-common-1.1.x.x | storm-common-1.1.x.x |
| storm-dynamic-info-provider-1.7.x.x | storm-dynamic-info-provider-1.7.4-3 | storm-dynamic-info-provider-1.7.4-3 |
| storm-front-end-server-1.7.x.x | storm-front-end-server-1.8.0-4 | storm-front-end-server-1.8.1-1 |
| storm-globus-gridftp-server-1.1.0.x | storm-globus-gridftp-server-1.2.0-5 | storm-globus-gridftp-server-1.2.0-5 |
| storm-gridftp-client-1.1.0-6 | storm-gridftp-client-1.1.0-6 | storm-gridftp-client-1.1.0-6 |
| storm-srm-client-1.5.0-x | storm-srm-client-1.6.0-6 | storm-srm-client-1.6.0-7 |
| yaim-storm-4.1.x-x | yaim-storm-4.2.x-x | yaim-storm-4.3.0-21 |
| VOMS     |       |       |       |
| voms-2.0.x.x | voms-2.0.x.x | voms-2.0.x.x |
| voms-admin-client-2.0.16-1 | voms-admin-client-2.0.17-1 | voms-admin-client-2.0.18-1 |
| voms-admin-server-2.6.1-1 | voms-admin-server-2.7.0-1 | voms-admin-server-2.7.1-1 |
| voms-api-java-2.0.x.x | voms-api-java-2.0.x.x | voms-api-java-3.0.x.x |
| voms-clients-2.0.x.x | voms-clients-2.0.8-1 | voms-clients-3.0.x.x |
| voms-devel-2.0.x.x | voms-devel-2.0.8-1 | voms-devel-3.0.x.x |
| voms-mysql-3.1.5-1 | voms-mysql-3.1.6-1 | voms-mysql-3.1.6-1 |
| voms-oracle-3.1.12-1 | voms-oracle-3.1.12-1 | voms-oracle-3.1.12-1 |
| voms-server-2.0.x.x | voms-server-2.0.8-1 | voms-server-2.0.8-1 |
| yaim-voms-1.x.x.x | yaim-voms-1.1.1 | yaim-voms-1.1.1 |
| WMS      |       |       |       |
| wms-broker-3.3.x-x | wms-broker-3.4.0-4 | wms-broker-3.4.0-4 |
| wms-broker-info-access-3.3.2-3 | wms-broker-info-access-3.4.0-4 | wms-broker-info-access-3.5.0-3 |
| wms-classad-plugin-3.3.1-3 | wms-classad-plugin-3.4.0-4 | wms-classad-plugin-3.4.0-4 |
| wms-config-configuration-3.3.x-x | wms-config-configuration-3.4.0-5 | wms-config-configuration-3.5.0-5 |
| wms-helper-3.3.x-x | wms-helper-3.4.0-5 | wms-helper-3.4.0-5 |
| wms-ice-3.3.x-x | wms-ice-3.4.0-7 | wms-ice-3.4.0-7 |
| wms-uaas-3.3.x-x | wms-uaas-3.4.0-7 | wms-uaas-3.4.0-7 |
| wms-jobsubmission-3.3.x-x | wms-jobsubmission-3.4.0-9 | wms-jobsubmission-3.5.0-9 |
| wms-manager-3.3.x-x | wms-manager-3.4.0-6 | wms-manager-3.4.0-6 |
| wms-matching-maker-3.3.x-x | wms-matching-maker-3.4.0-6 | wms-matching-maker-3.4.0-6 |
| wms-purger-3.3.x-x | wms-purger-3.4.0-4 | wms-purger-3.4.0-4 |
| wms-ua-api-python-3.3.x-x | wms-ua-api-python-3.4.0-5 | wms-ua-api-python-3.5.0-3 |
| wms-ua-commands-3.3.x-x | wms-ua-commands-3.4.0-5 | wms-ua-commands-3.5.0-5 |
| wms-ua-configuration-3.3.x.x | wms-ua-configuration-3.4.2-3 | wms-ua-configuration-3.5.0-5 |
| wms-utils-classad-3.2.2-2 | wms-utils-classad-3.3.0-2 | wms-utils-classad-3.4.0-4 |
| wms-utils-exception-3.2-2-2 | wms-utils-exception-3.3-0-2 | wms-utils-exception-3.4-0-4 |
| wms-wmp-proxy-api-cpp-3.3.x-x | wms-wmp-proxy-api-cpp-3.4.0-4 | wms-wmp-proxy-api-cpp-3.5.0-3 |
| wms-wmp-proxy-api-java-3.3.3-3 | wms-wmp-proxy-api-java-3.4.0-4 | wms-wmp-proxy-api-java-3.5.0-3 |
| wms-wmp-proxy-api-python-3.3.x-x | wms-wmp-proxy-api-python-3.4.0-4 | wms-wmp-proxy-api-python-3.5.0-3 |
| wms-wmp-proxy-interface-3.3.x-x | wms-wmp-proxy-interface-3.4.0-4 | wms-wmp-proxy-interface-3.5.0-3 |
| yaim-wms-4.1.x-x | yaim-wms-4.2.0-6 | yaim-wms-4.2.0-6 |
| WNoDeS   |       |       |       |
| wnodes-bait-2.0.x-x | wnodes-bait-2.0.8-3 | wnodes-bait-2.0.8-3 |
| wnodes-cachemanager-2.0.x-x | wnodes-cachemanager-2.0.13 | wnodes-cachemanager-2.0.13 |
| wnodes-cli-1.0.3-12 | wnodes-cli-1.0.3-12 | wnodes-cli-1.0.3-12 |
| wnodes-cloud-1.0.0-7 | wnodes-cloud-1.0.0-7 | wnodes-cloud-1.0.0-7 |
| wnodes-hypervisor-2.0.x-x | wnodes-hypervisor-2.0.5-9 | wnodes-hypervisor-2.0.5-9 |
| wnodes-manager-2.0.x-x | wnodes-manager-2.0.3-5 | wnodes-manager-2.0.3-5 |
| wnodes-namesserver-2.0.x-x | wnodes-namesserver-2.0.4-3 | wnodes-namesserver-2.0.4-3 |
| wnodes-site-specific-2.0.x-x | wnodes-site-specific-2.0.2-3 | wnodes-site-specific-2.0.2-3 |
| wnodes-utils-2.0.x-x | wnodes-utils-2.0.4-3 | wnodes-utils-2.0.4-3 |
| YAIM     |       |       |       |
| glite-yaim-client-5.0.0-1 | glite-yaim-client-5.0.1-2 | glite-yaim-client-5.2.0-1 |
| glite-yaim-core-5.0.0-1 | glite-yaim-core-5.1.0-1 | glite-yaim-core-5.1.2-1 |

The measurement of the selected metrics has been organized as follows. While for the
quality metric we considered the information contained in the release notes of each software products [17], and calculated NOD for each software product by hand; for the others we used the following open source tools. Cloc [18] counts blank lines, comment lines, and physical lines of source code in C, C++, Python, Java, Perl, Bourne Shell, C Shell, and other programming languages. Pmc McCabe [19] calculates McCabe-style cyclomatic complexity for code in C and C++. Radon [20] calculates various metrics from the source code such as McCabe’s cyclomatic complexity, SLOC, comment lines, and blank lines for source code in Python. Pylint [21] is an analyzer for code in Python, which looks for programming errors, helps enforcing a coding standard and sniffs for some code smells. Findbugs [22] performs static analysis for code in Java. JavaNCSS [23] measures McCabe-style cyclomatic complexity and source statements for code in Java.

With the collected data we performed a first analysis to identify those software products with some criticalities, such as the presence of anomalous behaviour or the contribution to system failures. Then, they determined the inputs of predictive models based on the discriminant analysis technique. With the produced outputs we settled the risk of these products to be fault-prone [24], [25], [26] and we compared it with the criticalities. We implemented a simple Matlab script to elaborate the input data conveniently and produced significative outcomes.

3. Raw Data Analysis

For each size metric we set a threshold over which software package might have issues about its maintainability, stability and usability: 10000 for LOC, 10000 for BLOC, 10000 for CLOC, 4 for NOPL, 4 for NOE, and 200 for NOF. We observed what follows for the software packages detailed in table 1: LOC - the glite-ce-cream-cli, voms, storm-frontend-server, and wms-ice software packages might be the most complicated packages to maintain due to the high number of code lines; BLOC and CLOC - the glite-ce-cream-client-api-c, voms-admin-server, storm-backend-server, and wms-common software packages might falsify the productivity level of the correspondent software product because of the high number of blank lines; NOPL and NOE - the storm-backend-server, voms, and glite-ce-cream-utils software packages might be ported on other platforms with difficulty containing at least four programming languages. The supported languages, such as C Shell, Bourne Shell, Python, Java, C++ and C, are distributed among the software packages and might contribute to a reduction in team effort for their maintainability; NOF - the glite-ce-crema-cli, voms, voms-admin-server, storm-backend-server, and storm-backend-frontend software packages might be maintained with difficulties over time due the the high number of files. For complexity metric we adopted the rank order of the score of block complexity as specified in the radon documentation [27]: 1-5 low simple block; 6-10 well-structured and stable block; 11-20 moderate-slightly complex block; 21-30 more than moderate-more complex block; 31-40 high-complex block, alarming; 41+ very high-error prone. According to this ranking we observed alarming block and error prone block scores in the majority of the software products with the exclusion of YAIM in each EMI distribution: CREAM, StoRM, VOMS and WMS show such issues for code in C and C++; WMS and WNoDeS for code in Python; CREAM, StoRM and VOMS for code in Java. Concerning the C and C++ code, the main cause is the inclusion of external software in the packages like the std2soap.c file; furthermore these types of blocks remain the same or increase over the EMI distributions. Table 2 shows the size metrics measured for all the software products in all the EMI distributions.

For the defect metric, we noticed that defects were related to code, build, package, and documentation. Table 4 shows the defect density of the EMI-th distribution as a non-linear combination of \( \frac{TNOD}{TLOC} \) over software products, where \( TNOD \) is the total number of defects and \( TLOC \) is the total lines of code.
### Table 2. Size metrics measured for all the software products in all the EMI distributions

| Size Metrics | EMI Distributions | Software Products |
|--------------|-------------------|-------------------|
|              | CREAM | StoRM | VOMS | WMS | WNoDeS | YAIM |
| Total NOF    |       |       |      |     |        |      |
| EMI 1        | 407   | 1063  | 2872 | 1269 | N.A.   | 31   |
| EMI 2        | 723   | 1304  | 1605 | 1229 | 104    | 31   |
| EMI 3        | 680   | 1521  | 1689 | 923  | 93     | 256  |
| Max NOPL     |       |       |      |     |        |      |
| EMI 1        | 4     | 4     | 6    | 4   | N.A.   | 3    |
| EMI 2        | 3     | 3     | 4    | 3   | 2      | 3    |
| Max NOE      |       |       |      |     |        |      |
| EMI 1        | 7     | 11    | 11   | 9   | N.A.   | 8    |
| EMI 2        | 7     | 11    | 10   | 8   | 5      | 8    |
| EMI 3        | 7     | 11    | 10   | 9   | 5      | 8    |
| Total LOC    |       |       |      |     |        |      |
| EMI 1        | 124860| 278625| 553930| 790149| N.A. | 2837 |
| EMI 2        | 151919| 393053| 465238| 737718| 38564| 2966 |
| EMI 3        | 73929 | 419897| 464051| 336486| 61461| 2966 |
| Total BLOC   |       |       |      |     |        |      |
| EMI 1        | 22471 | 44931 | 112453| 122407| N.A. | 736  |
| EMI 2        | 29298 | 59514 | 88011 | 109020| 6072 | 751  |
| EMI 3        | 18316 | 63020 | 52576 | 10505 | 11651| 1226 |
| Total CLOC   |       |       |      |     |        |      |
| EMI 1        | 24716 | 68378 | 105021| 160404| N.A. | 1201 |
| EMI 2        | 31170 | 85592 | 89299 | 154831| 7028 | 1223 |
| EMI 3        | 1182  | 6516  | 3724  | 2466  | 397  | 75   |
| Max N1-5     |       |       |      |     |        |      |
| EMI 1        | 1022  | 5616  | 3449  | 2124 | N.A.  | 55   |
| EMI 2        | 1128  | 5729  | 3685  | 2475  | 268  | 55   |
| EMI 3        | 1182  | 6516  | 3724  | 2466  | 397  | 75   |
| Max N6-10    |       |       |      |     |        |      |
| EMI 1        | 149   | 310   | 170   | 400   | N.A. | 10   |
| EMI 2        | 163   | 339   | 170   | 446   | 11   | 10   |
| EMI 3        | 178   | 416   | 168   | 450   | 12   | 10   |
| Max N11-20   |       |       |      |     |        |      |
| EMI 1        | 61    | 134   | 80    | 204   | N.A. | 2    |
| EMI 2        | 75    | 147   | 51    | 207   | 7    | 3    |
| EMI 3        | 79    | 171   | 53    | 208   | 7    | 9    |
| Max N21-30   |       |       |      |     |        |      |
| EMI 1        | 29    | 44    | 14    | 62    | N.A. | 0    |
| EMI 2        | 30    | 45    | 10    | 68    | 2    | 1    |
| EMI 3        | 33    | 45    | 9     | 85    | 2    | 4    |
| Max N31-40   |       |       |      |     |        |      |
| EMI 1        | 4     | 17    | 7     | 26    | N.A. | 0    |
| EMI 2        | 11    | 17    | 5     | 29    | 1    | 0    |
| EMI 3        | 11    | 24    | 5     | 61    | 2    | 2    |
| Max N41+     |       |       |      |     |        |      |
| EMI 1        | 7     | 4     | 25    | 30    | N.A. | 0    |
| EMI 2        | 14    | 4     | 16    | 31    | 2    | 0    |
| EMI 3        | 16    | 8     | 16    | 35    | 2    | 1    |

### 4. Statistical Evaluation

The same collected data determine the level of risk of a software product to be fault-prone as function of the level of importance of all the measured metrics. We call a product "fault-prone" if it is likely to contain a high number of faults [28] by analysing 62 software packages (shown in table 1) in all the three EMI distributions. This was done without setting an appropriate threshold on the number of faults expected because it represents an uncertain resource constraint at the time of modeling: therefore we only predicted the rank-order of software products [29].

By using statistical theory we calculated data to determine the level of importance of each metric ($MRL_j$) and the level of risk of each software product ($SPRL_z$). Considering all the software packages $p = 65$, we calculated $MRL_j$ of the $j$-th metric as equation 1, where the weight $R_i$ of the $i$-th package is multiplied over the normalized deviation of the $i$-th package; $R_i$ is given by the fraction between the total number of defects $d_i$ and the length of time period $t_i$ over which the defects occurred for a given package, while $x_{i,j}$ is the $j$-th metric measure of the $i$-th package. Considering all the metrics $m = 13$ and the software products $n = 6$ that contain a given set of software packages $[h, k]$ (as specified in table 1), we calculated $SPRL_z$ of the $z$-th product normalized considering its total lines of code $TLOC_z$ as equation 2, where $MRL_j$ is multiplied over $x_{i,j}$. Table 3 shows the set of critical metrics highlighting their minimum and maximum values: the higher the value of MRL for a metric, the higher is its importance. Table 5
shows the levels of software products highlighting their minimum values and maximum values over distributions: the higher the value of SPRL, the less is the risk of faults in the product.

\[ MRL_j = \sum_{i=1}^{p} R_i \cdot \frac{|x_{i,j} - \mu_j|}{\sigma_j} \]  

\[ SPRL_z = \frac{1}{TLOCz} p \sum_{i=1}^{k} (\sum_{j=1}^{m} MRL_j \cdot x_{i,j}) \]

Table 3. Levels of Importance of Each Metric MRLj.

| Metrics | EMI 1 | EMI 2 | EMI 3 |
|---------|-------|-------|-------|
| NOF     | 1.3362 | 0.7128 | 0.7155 |
| NOPL    | 1.8041 | 0.8449 | 0.9286 |
| NOE     | 1.8041 | 0.8449 | 0.9286 |
| LOC     | 1.5338 | 0.7913 | 0.9199 |
| BLOC    | 1.4750 | 0.8087 | 0.9197 |
| CLOC    | 1.3018 | 0.5460 | 0.6616 |
| N1-5    | 1.3171 | 0.6117 | 0.6646 |
| N6-10   | 1.5600 | 0.6506 | 0.7055 |
| N11-20  | 1.5849 | 0.6862 | 0.7303 |
| N21-30  | 1.4912 | 0.6765 | 0.5378 |
| N31-40  | 1.3744 | 0.5120 | 0.4994 |
| N41     | 1.3391 | 0.6788 | 0.7739 |
| NOD     | 2.0128 | 1.4964 | 0.7890 |

Table 4. Defects Density vs Software Product Size over Software Products.

Table 5. Predicted Defects for all the software products in all the EMI distributions

| Software Products | EMI Distributions | SPRL | Predicted Defects | Detected Defects |
|-------------------|-------------------|------|-------------------|------------------|
| CREAM             | EMI 1             | 0.1933 | 19                | 30               |
|                   | EMI 2             | 0.0997 | 23                | 62               |
|                   | EMI 3             | 0.1265 | 13                | 15               |
| StoRM             | EMI 1             | 0.3611 | 39                | 57               |
|                   | EMI 2             | 0.1774 | 53                | 17               |
|                   | EMI 3             | 0.2057 | 56                | 31               |
| VOMS              | EMI 1             | 0.1816 | 73                | 41               |
|                   | EMI 2             | 0.1499 | 62                | 21               |
|                   | EMI 3             | 0.1861 | 62                | 32               |
| WMS               | EMI 1             | 0.1273 | 102               | 70               |
|                   | EMI 2             | 0.0945 | 96                | 51               |
|                   | EMI 3             | 0.1488 | 46                | 16               |
| WNoDeS            | EMI 1             | N.A.   | N.A.              | N.A.             |
|                   | EMI 2             | 1.8079 | 9                 | 27               |
|                   | EMI 3             | 1.4052 | 11                | 19               |
| YAIM              | EMI 1             | 35.4611 | 4          | 25               |
|                   | EMI 2             | 23.5059 | 4          | 11               |
|                   | EMI 3             | 29.1175 | 4          | 2                |

At this time with the available data and the actual defect data, we used the most suitable method in seeking fault-prone software products [25] - the discriminant analysis that determines the minimum number of variables to discriminate the fault-prone products. As result, the statistical model with the specified method predicts the risk of a software product of being fault prone with a precision of 83%. Then, with the use of size and complexity metrics, we tried to predict the defects as shown in table 5. This time we used a regression predictive method [26] that ignores the underlying casual effects of programmers and designers - the human factors. As a consequence, this determines an inaccurate number of defects.
5. Conclusions
Considering the available data, we observed that predictive models may be valid instruments to determine the quality of software in the early phases of development to reconcile stability and quick defect fixing. We have shown that not all predictive methods actually work. The discriminant analysis method gives quite good results (83%) and is much better than the used regression method. The outcomes for all methods improve for large amount of data. Therefore future work should aim at verifying further predictive methods to identify those that provide better defect predictions.

References
[1] Musa J D 1998 Software Reliability Engineering (Osborne/McGraw-Hill)
[2] Rae A K, Hausen H L and Robert P 1994 Software Evaluation for Certification: Principles, Practice and Legal Liability (The McGraw-Hill International Software Quality Assurance Series) (Mcgraw Hill Book Co Ltd)
[3] Boehm B W, Brown J R and Lipow M 1976 Quantitative evaluation of software quality the 2nd International Conference on Software Engineering (Los Alamitos)
[4] Hall T, Beecham S, Hall T, Bowes D, Gray D and Counsell S 2012 IEEE Transaction on Software Engineering 36 1276–1304
[5] Nasa-softwaredefectdatasets URL http://nasa-softwaredefectdatasets.wikispaces.com
[6] Shepperd M, Song Q, Sun Z and Mair C 2013 IEEE Transaction on Software Engineering 39 1208–1215
[7] Home - european middleware initiative URL http://www.eu-emi.eu/
[8] Aiitiinieie C, Ceccanti A, Dongiovanni D, Meglio A D and Giacomini F 2012 Journal of Physics: Conference Series 396 URL http://iopscience.iop.org/1742-6596/396/5/052002
[9] Andreott P, Bertocco S, Capannini F, Cecchi M, Dorigo A, Frizziero E, Giaihelle A, Giacomini F, Mezzadri M, Monforte S, Prez F, Molinari E, Rebato D, Sgaravotto M and Zangrandol M 2011 Journal of Physics: Conference Series 331 URL http://iopscience.iop.org/1742-6596/331/6/062024
[10] Zappi R, Ronchieri E, Forl A and Ghiselli A 2011 An Efficient Grid Data Access with StoRM (Springer New York) chap VI Grid Middleware and Interoperability, pp 239–250 Data Driven e-Schience. Use Cases and Successful Applications of Distributed Computing Infrastructures (ISGC 2010)
[11] Ceccanti A, Ciashchini V, Dimou M, Garzoglio G, Levshina T, Traylen S and Venturi V 2009 Journal of Physics: Conference Series 219 URL http://iopscience.iop.org/1742-6596/219/6/062006
[12] Cecchi M, Capannini F, Dorigo A, Ghiselli A, Giacomini F, Maraschini A, Marzolla M, Monforte S, Pacini F, Petronzio L and Prez F 2009 The glite workload management system Advanced in Grid and Pervasive Computing (Lecture Notes in Computer Science vol 5529) (Springer Berlin Heidelberg) pp 256–268
[13] Salomoni D, Italiano A and Ronchieri E 2011 Journal of Physics: Conference Series 331 URL http://iopscience.iop.org/1742-6596/331/5/052017
[14] Ayala P M, Rajeswari S and Murty S A V S 2009 Application of yaim tool in grid computing Tech. rep. Superintendents Advisory Committee on Enrolment and Transfers (SACET)
[15] Kan S H 2002 Metrics and Models in Software Quality Engineering (Addison-Wesley Professional)
[16] McCabe T J 1976 IEEE Transactions on Software Engineering SE-2 308–320
[17] Releases-european middleware initiative URL www.eu-emi.eu/releases
[18] Cloc-count lines of code URL http://cloc.sourceforge.net
[19] pmccabe package: Ubuntu URL https://launchpad.net/ubuntu/+source/pmccabe
[20] radon 0.4.3. Python package index URL https://pypi.python.org/pypi/radon
[21] Pylint user manual URL http://docs.pylint.org/
[22] findbugs-static analysis tool to find coding defects in java programming URL code.google.com/p/findbugs/
[23] Javanccs-a source measurement suite for java URL www.kclee.de/clemens/java/javanccs
[24] Zheng J, Williams L, Nagappan N and Snipes W 2006 IEEE Transaction on Software Engineering 32 240–253
[25] Guo G and Guo P 2008 Experimental study of discriminant method with application to fault-prone module detection International Conference on Computational Intelligence and Security
[26] Fenton N 1990 Journal of Software Engineering 565–78
[27] Using radon programmatically - radon 0.4.3 documentation URL https://radon.readthedocs.org/en/latest/api.html
[28] Ohlsson N and Wohlin C 1996 Identification of failure-prone modules in two software system releases 21st Annual Software Engineering Workshop (Greenbelt, Maryland, USA)
[29] Khoshgoftaar T M and Seliya N 2003 Empirical Software Engineering Journal 8 325–350