Evaluating Predictive Models of Software Quality

Vincenzo Ciaschini, Marco Canaparo, Elisabetta Ronchieri, Davide Salomoni

INFN CNAF, Bologna, Italy

CHEP 2013, October 14 - 18, 2013, Amsterdam, The Netherlands

Motivation

- Can we predict whether our products and processes will meet goals for quality during the development life cycle?

- Specifically, can we determine <u>the risk factor</u> associated to our products in relation to reliability and maintainability in order to prevent faults?

Why? Motivation again

- To produce better software at lower cost with predictable resource allocations and time estimations.
- To detect defects from faults early in order to mi their consequences.
- To estimate time to the products’ deployment.
- To predict and improve quality of the products and the development process.

What? Background

- The predictive model of software quality determines software quality level periodically and indicates software problems early.
- Over the last few decades several predictive methods have been used in the development of fault predictive models:
  - Regression;
  - Statistical;
  - Machine learning;
  - Others.

Context? Background

- The main context of these models is closed to NASA’s based projects: however open source systems are also considered.
- The C/C++ language dominates in the studies of these models [1]:
  - Over half of the models are built by analysing C/C++ code;
  - 20% of models are for Java code.

Measure of performances? Background

- In case of continuous output data, the performance of models is based on:
  - Error rates such as mean square error, mean absolute error, standard error, and absolute error;
  - Regression coefficients such as regression R2 (linear), cubic R2, and regression R2 (non-linear);
  - Correlation test such as Pearson and Spearman;
  - Variance significance test such as goodness-of-fit, Chi-Square and p-value.

Experiment Description

- The experiment consists of evaluating software quality of some EMI products by using predictive models:
  - As first approximation, we have selected the INFN software products of the EMI distributions [2] (i.e. EMI 1, EMI 2, and EMI 3), such as CREAM, StoRM, VOMS, WMS, WNoDeS, and parts of YAIM;
  - We have measured some static metrics [3] such as N. Files, N. Comments, N. Code, N. Languages, N. Blanks, and McCabe, for all the software products in each EMI distribution;
  - We have used open source tools to measure the metrics such as cloc [4], pmccabe [5] and radon [6];
  - We have collected defects from the release notes of each software products [7];
  - We have used statistical predictive model based on the discriminant analysis [8], [9], [10].

Software Packages

'x' means that the specified source package in a EMI distribution has been updated.

Products	EMI1 base/updates	EMI2 base/updates	EMI3 base/updates
CREAM	glite-ce-cream-1.13.x-x glite-ce-cream-api-java-1.13.x-x glite-ce-cream-cli-1.13.x-x glite-ce-cream-client-api-c-1.13.2-3 glite-ce-cream-utils-1.1.0-3 glite-ce-yaim-cream-ce-4.2.x-x	glite-ce-cream-1.14.x-x glite-ce-cream-api-java-1.14.x-x glite-ce-cream-cli-1.14.x-x glite-ce-cream-client-api-c-1.14.x-x glite-ce-cream-utils-1.2.x-x glite-ce-yaim-cream-ce-4.3.x-x	glite-ce-cream-1.x.x-x glite-ce-cream-api-java-1.x.x-x glite-ce-cream-cli-1.15.x-x glite-ce-cream-client-api-c-1.15.x-x glite-ce-cream-utils-1.3.x-x glite-ce-yaim-cream-ce-4.4.x-x
VOMS	voms-2.0.x-x voms-admin-client-2.0.16-1 voms-admin-server-2.6.1-1 voms-api-java-2.0.x-x voms-clients-2.0.x-x voms-devel-2.0.x-x voms-mysql-3.1.5-1 voms-oracle-3.1.12-1 voms-server-2.0.x-x yaim-voms-1.x.x-x	voms-2.0.x-x voms-admin-client-2.0.17-1 voms-admin-server-2.7.0-1 voms-api-java-2.0.x-x voms-clients-2.0.8-1 voms-devel-2.0.8-1 voms-mysql-3.1.6-1 voms-oracle-3.1.12-1 voms-server-2.0.8-1 yaim-voms-1.1.1-1	voms-2.0.x-x voms-admin-client-x.x.x-x voms-admin-server-3.0.x-x voms-api-java-3.0.x-x voms-clients-3.0.x-x voms-devel-2.0.8-1 voms-mysql-3.1.6-1 voms-oracle-3.1.15-2 voms-server-2.0.8-1 yaim-voms-1.1.1-1
StoRM	storm-backend-server-1.x.x-x storm-common-1.1.x-x storm-dynamic-info-provider-1.7.x-x storm-frontend-server-1.7.x-x storm-globus-gridftp-server-1.1.0-x storm-srm-client-1.5.0-x yaim-storm-4.1.x-x	storm-backend-server-1.x.x-x storm-dynamic-info-provider-1.7.4-3 storm-frontend-server-1.8.0-x storm-globus-gridftp-server-1.2.0-4 storm-gridhttps-plugin-1.0.3-x storm-gridhttps-server-1.1.0-3 storm-pre-assembled-configuration-1.0.0-6 storm-srm-client-1.6.0-6 tstorm-1.2.1-2 yaim-storm-4.2.x-x	storm-backend-server-1.11.0-43 storm-dynamic-info-provider-1.7.4-4 storm-frontend-server-1.8.1-1 storm-globus-gridftp-server-1.2.0-5 storm-gridhttps-plugin-1.1.0-4 storm-gridhttps-server-2.0.0-230 storm-pre-assembled-configuration-1.1.0-8 storm-srm-client-1.6.0-7 tstorm-2.0.1-13 yaim-storm-4.3.0-21
YAIM	glite-yaim-clients-5.0.0-1 glite-yaim-core-5.0.0-1	glite-yaim-clients-5.0.1-2 glite-yaim-core-5.1.0-1	glite-yaim-clients-5.2.0-1 glite-yaim-core-5.1.2-1

Software Packages

'x' means that the specified source package in a EMI distribution has been updated.

Products	EMI1 base/updates	EMI2 base/updates	EMI3 base/updates
WMS	wms-broker-3.3.x-x wms-brokerinfo-3.3.1-3 wms-brokerinfo-access-3.3.2-3 wms-classad-plugin-3.3.1-3 wms-common-3.3.x-x wms-configuration-3.3.x-x wms-helper-3.3.x-x wms-ice-3.3.x-x wms-ism-3.3.x-x wms-jobsubmission-3.3.x-x wms-manager-3.3.x-x wms-matchmaking-3.3.x-x wms-purger-3.3.x-x wms-ui-api-python-3.3.3-3 wms-ui-commands-3.3.3-3 wms-ui-configuration-3.3.2-3 wms-utils-classad-3.2.2-2 wms-utils-exception-3.2.2-2 wms-wmproxy-3.3.x-x wms-wmproxy-api-cpp-3.3.3-3 wms-wmproxy-api-java-3.3.3-3 wms-wmproxy-api-python-3.3.3-3 wms-wmproxy-interface-3.3.3-3 yaim-wms-4.1.x-x	wms-broker-3.4.0-4 wms-brokerinfo-3.4.0-4 wms-brokerinfo-access-3.4.0-4 wms-classad-plugin-3.4.0-4 wms-common-3.4.0-5 wms-configuration-3.4.0-5 wms-helper-3.4.0-5 wms-ice-3.4.0-7 wms-ism-3.4.0-7 wms-jobsubmission-3.4.0-9 wms-manager-3.4.0-6 wms-matchmaking-3.4.0-6 wms-purger-3.4.0-4 wms-ui-api-python-3.4.0-5 wms-ui-commands-3.4.0-x wms-ui-configuration-3.3.2-3 wms-utils-classad-3.3.0-2 wms-utils-exception-3.3.0-2 wms-wmproxy-3.4.0-7 wms-wmproxy-api-cpp-3.4.0-4 wms-wmproxy-api-java-3.4.0-4 wms-wmproxy-api-python-3.4.0-4 wms-wmproxy-interface-3.4.0-x yaim-wms-4.2.0-6	wms-brokerinfo-access-3.5.0-3 wms-common-3.x.x-x wms-configuration-3.x.x-x wms-core-3.5.0-7 wms-ice-3.5.0-4 wms-interface-3.x.x-x wms-jobsubmission-3.5.0-3 wms-purger-3.5.0-3 wms-ui-api-python-3.5.0-3 wms-ui-commands-3.5.x-x wms-utils-classad-3.4.x-x wms-utils-exception-3.4.x-x wms-wmproxy-api-cpp-3.5.0-3 yaim-wms-4.2.0-6
WNoDeS	-	wnodes-bait-2.0.x-x wnodes-hypervisor-2.0.x-x wnodes-manager-2.0.x-x wnodes-nameserver-2.0.x-x wnodes-site-specific-2.0.x-x wnodes-utils-2.0.x-x	wnodes-accounting-1.0.0-4 wnodes-bait-2.0.8-3 wnodes-cachemanager-2.0.1-3 wnode-cli-1.0.3-12 wnodes-cloud-1.0.0-7 wnodes-hypervisor-2.0.5-9 wnodes-manager-2.0.3-5 wnodes-nameserver-2.0.4-3 wnodes-site-specific-2.0.2-3 wnodes-utils-2.0.4-3

Metrics

Size Category	Descriptions
N. Files	Determines the number of files in a software package;
N. Blank	Determines the number of blank lines found in the files of the software package.
N. Comments	Determines the number of comment lines found in the files of the software package;
N. Code	Indicates the number of code lines found in the files of the software package. A very high count might indicate that a type or method might be hard to maintain.
N. Languages	Determines the number of programming languages supported in the software package;
N. Extensions	Determines the number of extensions found in the software package.

DIRECT MEASURES

Complexity Category	Descriptions
McCabe cyclomatic complexity	Determines the complexity of a section of source code by measuring the number of linearly independent paths in the flow of the source code. A complex control flow will require more tests to achieve good code coverage and will penalize its maintanability.

Quality Category	Descriptions
Defects	Determines the reported defects calculated at the end of each release.

INDIRECT MEASURES

Metrics Tools

Names	Descriptions	Languages
cloc	counts blank lines, comment lines, and physical lines of source code.	C, C++, Python, Java, Perl, Bourne Shell, C Shell, etc
pmccabe	calculates McCabe-style cyclomatic complexity for C and C++ source code.	C, C++
radon	calculates various metrics from the source code such as McCabe's cyclomatic complexity, raw metrics (such as SLOC, comment lines, and blank lines), Halstead metrics, and Maintainability Index.	python
pylint	checks that a module satisfy a coding standard, detects duplicated code and other more.	python
findbugs	identifies bug patterns.	Java
javancss	measures two standard metrics: McCabe-style cyclomatic complexity and source statements.	Java

Size Metrics' Measures Interpretation

- Per software products (CREAM, VOMS, StoRM, WMS, WNoDeS, YAIM) in each EMI distribution:

$$Tot.N.Files(EMIDistribution, SoftwarePackage) = \begin{bmatrix} 407 & 2872 & 1063 & 1269 & 0 & 31 \\ 723 & 1605 & 1034 & 1229 & 104 & 31 \\ 680 & 1699 & 1521 & 923 & 256 & 31 \end{bmatrix}$$

$$Tot.N.Code(EMIDistribution, SoftwarePackage) = \begin{bmatrix} 124806 & 553930 & 278625 & 790149 & 0 & 2837 \\ 151919 & 465238 & 393053 & 737718 & 38564 & 2966 \\ 73929 & 464051 & 419887 & 336486 & 61461 & 2966 \end{bmatrix}$$

- The following considerations are per software packages in each distribution.

N. Code ≥ 10k	N. Blanks ≥ 10k	N. Comments ≥ 10k	N. Languages ≥ 4	N. Files > 200
glite-ce-cream-cli, voms, storm-frontend-server, and wms-ice might be the most complicated packages to maintain due to the high number of code lines.	glite-ce-cream-client-api-c, voms-admin-server, storm-backend-server, and wms-common might falsify the productivity level of the correspondent software product because of the high number of blank lines.	glite-ce-cream-client-api-c, voms-admin-server, storm-backend-server, and wms-common might falsify the productivity level of the correspondent software product because of the high number of comment lines.	storm-backend-server, voms, and glite-ce-cream-utils 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 in reducing team effort for their maintainability.	glite-ce-crema-cli, voms, voms-admin-server, storm-backend-server, and storm-backend-frontend might be maintained with difficulties over time due the the high number of files.

McCabe Complexity Metric's Measures Interpretation

- The measure of this metric provides for each block the score of complexity ranked as follows [6]:
  - 1-5: low- simple block;
  - 6-10 : low-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.
- The following considerations are for software products in each distribution:

Issues	C++/C	Python	Java
Alarming Blocks	CREAM, VOMS, StoRM, WMS	WMS, WNoDeS	CREAM, VOMS, StoRM
Error Prone Blocks	CREAM, VOMS, StoRM, WMS	WMS, WNoDeS	CREAM, VOMS, StoRM

- Concerning the C/C++ code:
  - the main cause is the inclusion of external software in the package like std2soap.c file;
  - furthermore these types of blocks remain constant or increase over the EMI distributions.

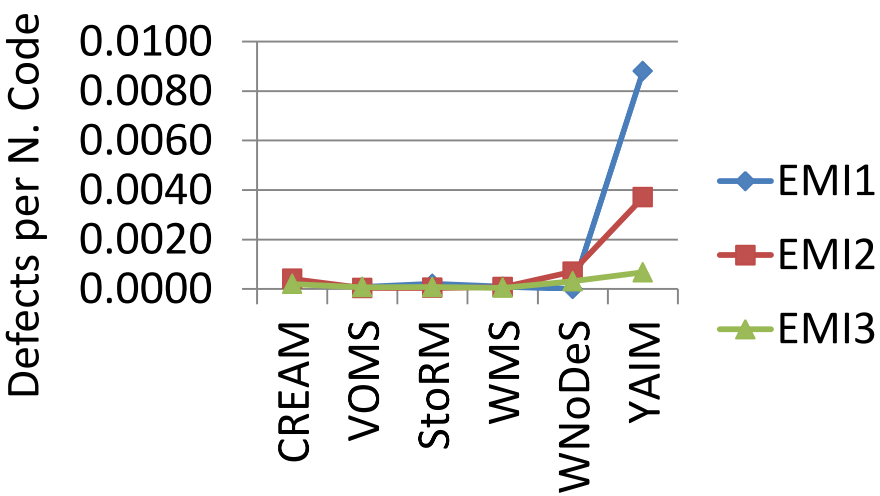
INFN Quality Measures: Defects

- Per software products (CREAM, VOMS, StoRM, WMS, WNoDeS, YAIM) in each EMI distribution:

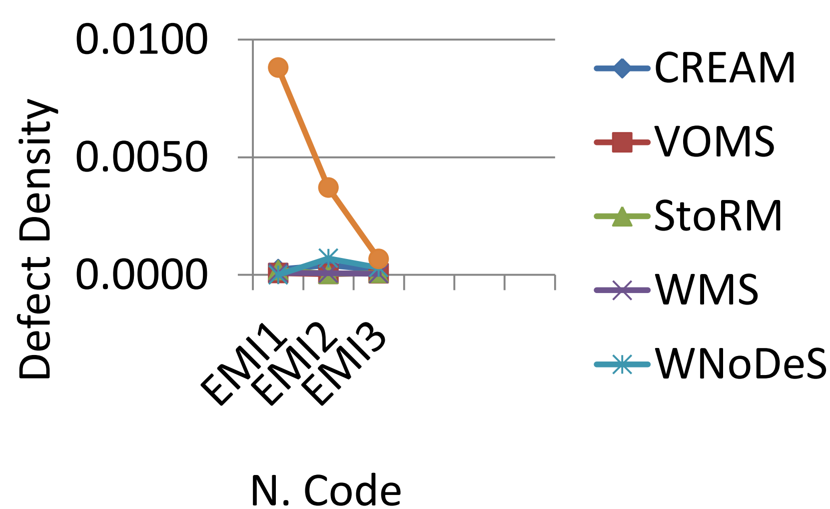
$$Tot.N.Defects(EMIDistribution, SoftwarePackage) = \begin{bmatrix} 30 & 41 & 57 & 70 & 0 & 25 \\ 62 & 21 & 17 & 51 & 27 & 11 \\ 15 & 32 & 31 & 16 & 19 & 2 \end{bmatrix}$$

- Defects decreased over the EMI distributions with the exclusion of VOMS and StoRM products.
- They were related to code, build and package, documentation.

Defect Density vs software product size



Defect Density = N. Defects / N. Code



Statistical Evaluation

- Considering all the software products (i.e. CREAM, VOMS, StoRM, WMS, WNoDeS, and YAIM) and the collected data for size, complexity and quality metrics, for each distribution firstly:
  - we have determined the level of risk/importance of each metric, and the level of risk of each software product to be fault prone by considering the discriminant analysis method that is the most suitable method in finding fault prone software products [9];
  - we have predicted the defects by using size and complexity metrics [10].
- Secondly we have evaluated the impact of this information in the EMI distributions.

Statistical Evaluation

The minimum value (red)

The maximum value (green)

Metrics	Level of Risk EMI1	EMI2	EMI3
N. Files	**1.4982**	1.3285	1.0588
N. Comments	2.0343	1.8043	1.0915
N. Blanks	2.0051	1.7247	1.1083
N. Code	2.0031	1.8014	**1.1446**
N. Extensions	**2.0969**	1.7565	0.7969
N. Languages	2.0794	1.7626	0.7516
McCabe: 1-5	1.9479	1.5444	0.7223
McCabe: 6-10	1.9686	1.5924	0.6522
McCabe: 11-20	2.0065	1.5401	**0.5921**
McCabe: 21-30	2.0135	**1.2134**	0.9339
McCabe: 31-40	1.9202	1.9730	0.7519
McCabe: 41+	1.8569	1.5825	1.0012
N. Defects	1.8158	**1.9910**	0.9081

Software Products	Is Fault Prone? EMI1	EMI2	EMI3
CREAM	$0.3474*10^6$	$0.3833*10^6$	$0.12830*10^6$
VOMS	$1.5559*10^6$	$1.1552*10^6$	**$0.73054*10^6$**
StoRM	$0.7913*10^6$	$0.9699*10^6$	$0.64969*10^6$
WMS	**$2.1616*10^6$**	**$1.8032*10^6$**	$0.55260*10^6$
WNoDeS	N.A.	$0.0937*10^6$	$0.09552*10^6$
YAIM	**$0.0099*10^6$**	**$0.0090*10^6$**	**$0.00566*10^6$**

Software Products	Predicted Defects EMI1	EMI2	EMI3
CREAM	19	23	13
VOMS	73	62	**62**
StoRM	39	53	56
WMS	**102**	**96**	46
WNoDeS	0	9	11
YAIM	**4**	**4**	**4**

Statistical Evaluation

Parameters	CREAM EMI1	EMI2	EMI3
Level of Risk	$0.3474*10^6$	$0.3833*10^6$	$0.1283*10^6$
Predicted Defects	19	23	13
Detected Defects	30	62	15

Parameters	WNoDeS EMI1	EMI2	EMI3
Level of Risk	0	$0.0937*10^6$	$0.0955*10^6$
Predicted Defects	0	9	11
Detected Defects	0	27	19

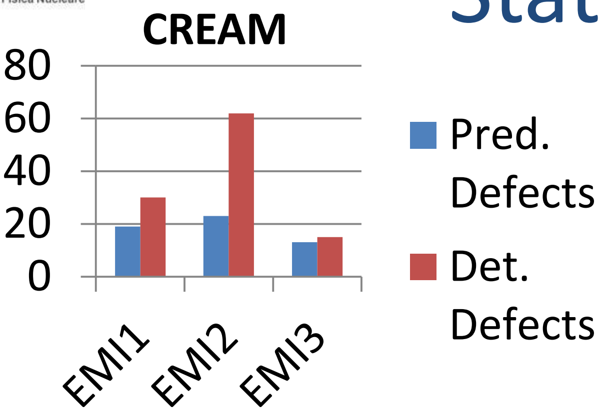
Parameters	StoRM EMI1	EMI2	EMI3
Level of Risk	$0.7913*10^6$	$0.9699*10^6$	$0.6497*10^6$
Predicted Defects	39	53	56
Detected Defects	57	17	31

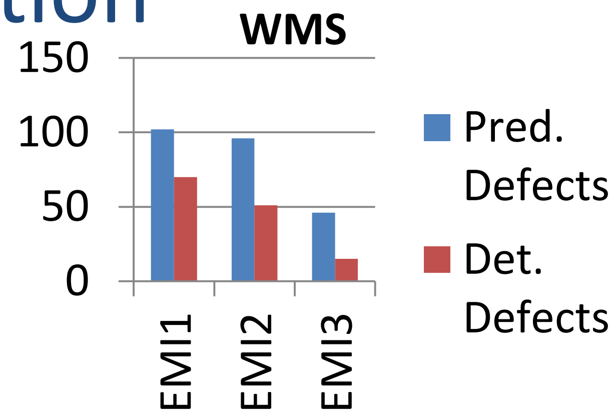
Parameters	YAIM EMI1	EMI2	EMI3
Level of Risk	$0.0099*10^6$	$0.0090*10^6$	$0.0057*10^6$
Predicted Defects	**4**	**4**	4
Detected Defects	**25**	**11**	2

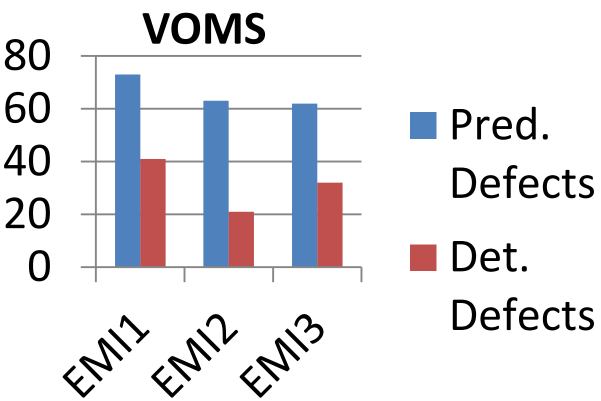
Parameters	VOMS EMI1	EMI2	EMI3
Level of Risk	$1.5559*10^6$	$1.1552*10^6$	$0.7305*10^6$
Predicted Defects	73	62	62
Detected Defects	41	21	32

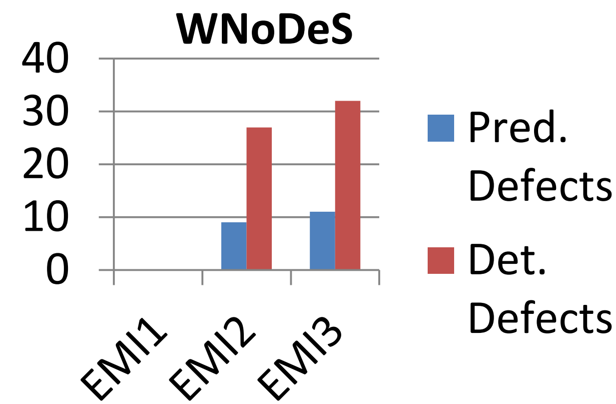
Parameters	WMS EMI1	EMI2	EMI3
Level of Risk	$2.1616*10^6$	$1.8032*10^6$	$0.5526*10^6$
Predicted Defects	**102**	**96**	46
Detected Defects	**70**	**51**	16

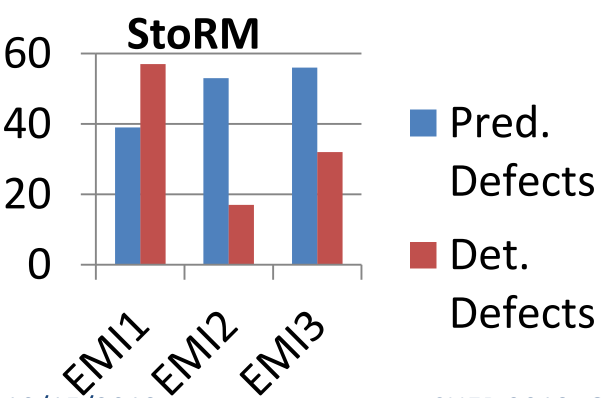
Statistical Evaluation

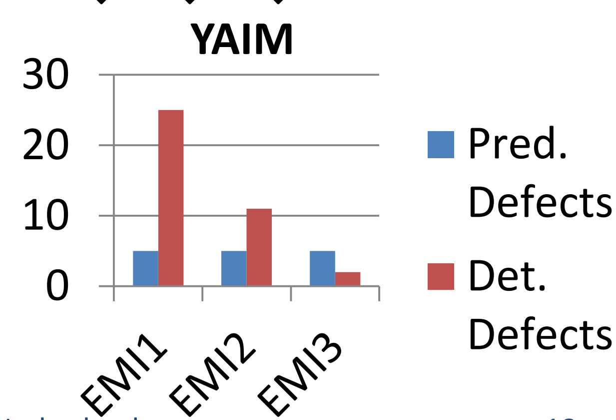












Conclusions

- Considering the available data and the detected defects the statistical model with the discriminant analysis method predicted the risk of being fault prone with a precision of 83%. This does not translate to precision in determining the number of defects, that was indeed wildly inaccurate.
- Their inputs are metrics’ measures that can come from existing software.
- Their precisions improve with the amount of data available.
- The above result shows that the effort necessary to learn this model will be repaid during the testing and quality assurance phase by suggesting which modules are more error prone and therefore should receive greater attention.

References

[1] Sara Beecham, Tracy Hall, David Bowes, David Gray, Steve Counsell, Sue Black, "A Systematic Review of Fault Prediction approaches used in Software Engineering", Lero Technical Report Lero-TR-2010-04.

[2] Stephen H. Kan, "Metrics and Models in Software Quality Engineering", Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA 2002

[3] Cristina Aiftimiei, Andrea Ceccanti, Danilo Dongiovanni, Andrea Di Meglio, Francesco Giacomini, "Improving the quality of EMI Releases by leveraging the EMI Testing Infrastructure," 2012 Journal of Physics: Conference Series Volume 396 Part 5.

[4] CLOC – Count Lines of Code, http://cloc.sourceforge.net.

[5] "pmccabe" package: Ubuntu, https://launchpad.net/ubuntu/+source/pmccabe.

[6] radon 0.4.3: Python Package Index, https://pypi.python.org/pypi/radon.

[7] Releases – European Middleware Initiative, www.eu-emi.eu/release.

[8] Jiang Zheng, Laurie Williams, Nachiappan Nagappan, Will Snipes, John P. Hudepohl, Mladen A. Vouk, "On the value of static analysis for fault detection in software", IEEE Transaction on Software Engineering, Vol. 32, No. 4, April 2006.

[9] Gege Guo, and Ping Guo, "Experimental Study of Discriminant Method with Application to Fault-Prone Module Detection", Proc. Of 2008 International Conference on Computational Intelligence and Security, December 2008.

[10] Norman Fenton, Paul Krause and Martin Neil, "A Probabilistic Model for Software Defect Prediction", IEEE Transaction on Software Engineering, 2001.