Experience with software quality metrics: two or three case studies

E. Ronchieri[1]

[1] INFN CNAF, Bologna, Italy

Workshop CCR 2015, Frascati, May 25-29, 2015

Contents

Research challenge

Research questions

- What software metrics are valuable to measure open software used in HEP research?
- What are the recommended values for each metric?
- Is there a way to predict fault components?

Context

- Use of large and mature research software
- Use of software metrics tools
- Use of statistical analysis
- Use predictive techniques

What observed

Aware of the experience acquired at INFN CNAF

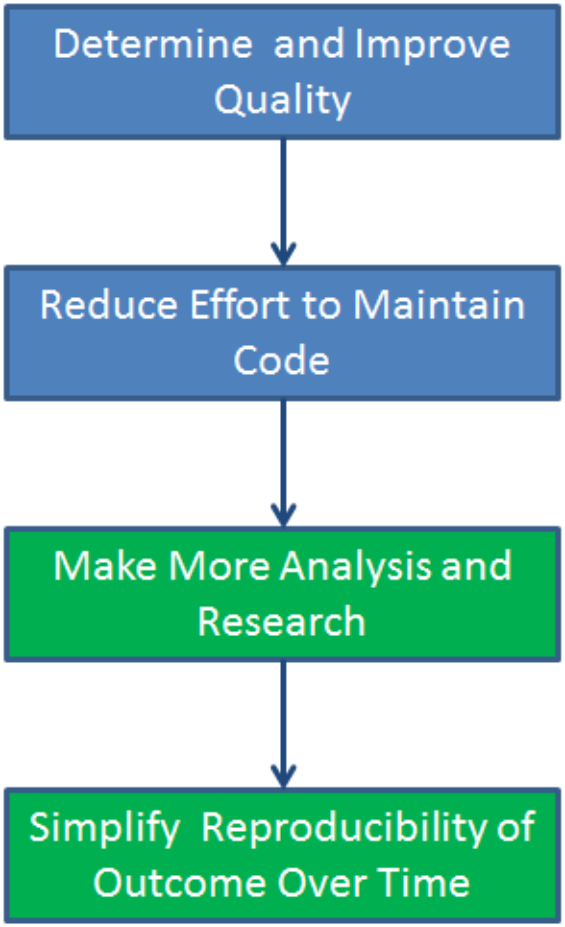
- Software researchers tend to neglect the quality of their products for different reasons:
  - developers distrust data from existing quality tools since they provide partial analysis of their software;
  - developers trust their own experience.

As a consequence, this has led to

- spend effort maintaining software once released;
- develop software without exploiting solutions for managing defects effectively.

What is achievable with quality

- Enhancing quality allows: reducing defects; saving costs; decreasing delivery delays.
- Furthermore, software quality models and metrics represent the mainstream to reach high reliability by balancing effort and results.



Quality

Many views of software quality

- The IEEE defines quality as the degree to which a system, component, or process meets specified requirements or customer or user needs or expectations.
- The International Organization for Standardization (ISO) defines quality as the degree to which a set of inherent characteristics fulfils requirements.
- Other experts define quality based on: conformance to requirements; fitness for use.
- However, a good definition must let us measure quality meaningfully.

Metric

- Metric is a quantitative measure of the degree to which a system, component or process possesses a given software attribute related to quality factors (also told characteristics).
- Metric allows us to:
  - estimate the cost and schedule of future projects;
  - evaluate the productivity impacts of new tools and techniques;
  - establish productivity trends over time;
  - improve software quality;
  - anticipate and reduce future maintenance needs.

Metrics Classification

Product metrics (are considered)

- Describe the characteristics of the product;
- Include size, complexity, performance and quality level.

Process metrics

- Improve software development and maintenance;
- Include defect removal, response time of the fix.

Project metrics:

- Describe project characteristics and execution;
- Include cost, N. developers, schedule and productivity.

Software quality standards

- Software quality standards describe software quality models categorizing software quality into a set of characteristics.
- About software attributes, ISO/IEC 25010:2011 standard defines 6 software factors, each subdivided in sub-characteristics (or criteria).
- About software guidelines, CMMI (Capability Maturity Model Integration) specification provides, amongst the others, the best practices definition.

Research methodology

Four stages

1. Used CMMI to identify best practice guidelines;
2. Used existing standard, called ISO/IEC 25010:2011 (former ISO/IEC 9126), to identify software maintainability;
3. Identified and evaluated software metrics tools (such as Imagix4D, SourceMonitor, LocMetrics, ACDC-Metrics, Understand) to collect a large amount of measurements of software characteristics;
4. Exploited a set of product metrics to assess the code state.

Stage 1: CMMI

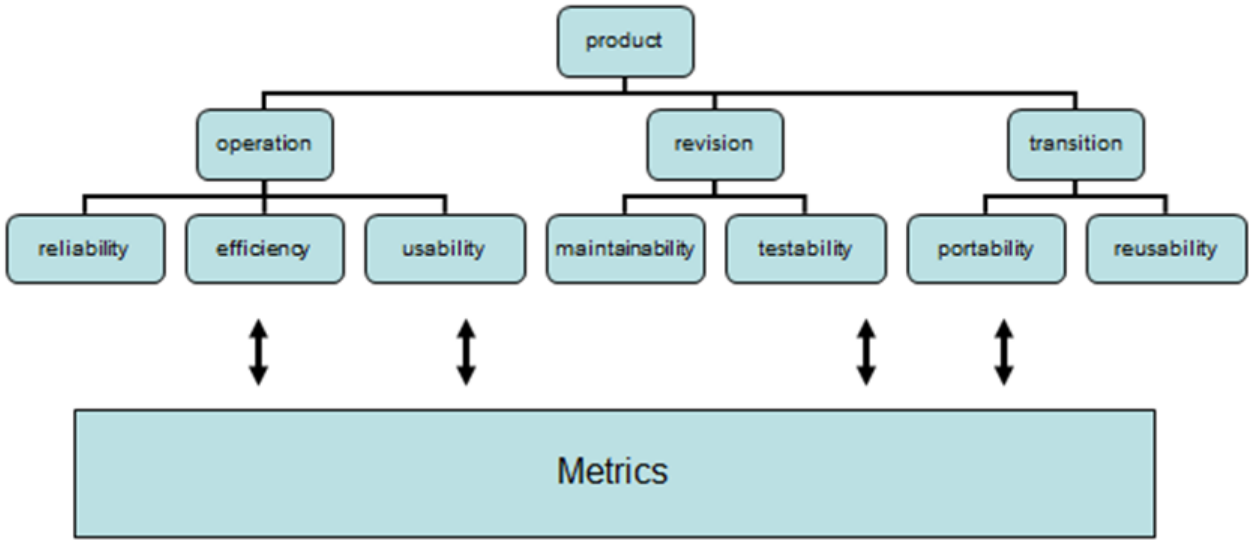
Best practices

- Software structure places a role since the initial stages of an application development;
- Configuration management is concerned not only with knowing and managing the state of all artefacts, but also with releasing distinct version of the system;
- Code construction involves accurate description of the software;
- Testing is an integral part of the software development;
- Deployment is the final stage of a system release.

Stage 2: ISO/IEC 25010:2011

Quality factors

- They include Functionality, Reliability, Usability, Efficiency, Maintainability and Portability
- But Maintainability is considered!



Stage 2: maintainability details

- Maintainability is for IEEE the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.

Sub-characteristics	Description
Analyzability	Characterizes the ability to identify the root cause of a failure within the software
Changeability	Characterizes the amount of effort to change a system
Stability	Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes
Testability	Characterizes the effort needed to verify (test) a system change

Stage 3: software metrics tools

Several commercial and free tools

Tool	Version	Description	Source	State
CCCC	3.1.4	analyzes C and C++ files and generates reports on various metrics of the code	open	Non Supported
CLOC	1.60	counts blank lines, comment lines and physical lines of source code	open	Supported
Imagix 4D	8.0.4	analyzes, documents and improves complex, third party or legacy C, C++ and Java software	under evaluation license	Supported
Pmccabe	2.6	calculates McCabe cyclomatic complexity and non-commented lines of code for C and C++ code	open	Non Supported
SLOCCount	2.26	computes Source Lines of Code	open	Supported
SourceMonitor	3.5.0.306	is a source code metrics measurement tool	open	Supported
Understand	3.1.278	is a static analysis tool	under evaluation license	Supported

Stage 3: encountered difficulties

- Some software metrics are not unequivocally defined
- Different interpretations of the same metric definition exist
  - Therefore there are different implementations of the same metric
- These tools produce non-comparable values
- They also use different names to refer to the same metric.
- Typically, these tools provide a subset of metrics with respect to the software factor they address.
  - Therefore to perform a comprehensive analysis, it is necessary to use several tools

Stage 3: encountered difficulties

- The metrics thresholds adopted in these tools are not always well-documented, making it difficult to understand what the tool wants to express
  - Existing thresholds are often based on researchers' experience and pertinent to the context of the considered software.
  - They must be re-evaluated in different context.

Stage 4: software product metrics

Grouped according to

- File
- Class
- Function

Categorized in

- Size
- Complexity
  - McCabe
  - Halstead
- Object-Orientation
  - Chidamber and Kemerer (CK)
- Others

Stage 4: main size metrics

Group	Size Metric	Source
File	Comment Ratio (CR_{File})	Lorenz
	Declarations in File (NOD_{File})	Lorenz
	File Size (bytes)	Lorentz
	Functions in File (NOF_{File})	Lorenz
	Lines in File ($TLOC_{File}$)	Lorenz
	Lines of Source Code ($SLOC_{File}$)	Lorenz
	Lines of Comments ($CLOC_{File}$)	Lorenz
	Number of Statements (NOS_{File})	Lorenz
	Variables in File (NOV_{File})	Lorenz
Function (F)	Lines in Function ($TLOC_F$)	Lorenz
	Lines of Source Code ($SLOC_F$)	Lorenz
	Variables in Function (NOV_F)	Lorenz

Comment Ratio represents the ratio of the lines of comments to the lines of source code in the file.

NOD is the number of top-level declarations in the file, including types, variables, functions and macro defines.

The definitions of the other metrics are evident.

Stage 4: main object-oriented metrics

Group	Object-Oriented Metric	Source
Class	Class Cohesion (LCOM)	CK
	Class Coupling (CBO)	CK
	Depth of Inheritance (DIT)	CK
	Number of Children (NOC)	CK
	Response for Class (RFC)	CK
	Weighted Methods (WMC)	CK

Lack of Cohesion of Methods (LCOM) is a measure of the cohesion of the member functions of the class.

Coupling Between Object (CBO) measures the coupling, or dependency, of the class.

DIT measures the depth of the hierarchy of base classes of the class.

NOC provides the number of classes directly derived from class.

RFC measures the number of methods called by the class methods.

WMC provides the total cyclomatic complexity for the class methods.

Stage 4: main complexity metrics

Group	Complexity Metric	Source
File, Function, Class	Intelligent Content (HI)	Halstead
	Mental Effort (HE)	Halstead
	Program Volume (HV)	Halstead
	Program Difficulty (HD)	Halstead
File, Class	Average Cyclomatic Complexity (MACC)	McCabe
	Maximum Cyclomatic Complexity (MMCC)	McCabe
	Total Cyclomatic Complexity (MTCC)	McCabe
File	Maintainability Index (MI)	Welker

HI measures the amount of content (complexity) of the file/function/class.

HE measures the number of elemental mental discriminations necessary to create, or understand, the file/function/class.

HV measures the information content of the file/function/class.

HD measures how compactly the file/function/class implements its algorithms.

MACC, MMCC and MTCC measures the average, maximum and total cyclomatic complexity for all methods in file/class.

MI measures the maintainability of the file, incorporating source code metrics into a single number.

Data Analysis Methodology

Four stages

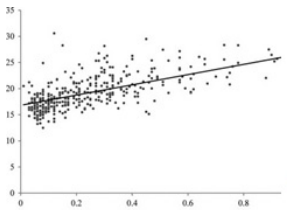
1. Use descriptive statistics for each release to get the distribution (mean and median), variance (standard deviation) and quantiles of each measure;
2. Adopt correlations between metrics to eliminate metrics that do not provide additional insights;
3. Identify thresholds from metrics analysis;
4. Use predictive techniques.

Use case 1

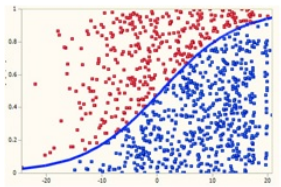
- To detect fault-prone and non fault-prone software components by using a new quality model:
  - that connects software best practices with a set of metrics (both static and dynamic metrics)
  - that uses predictive techniques, such as discriminant analysis and regression, to determine to what extent which metrics can influence a software component.
- The model was validated by using EMI products, such as CREAM, StoRM, VOMS, WNoDeS and WMS.

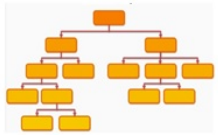
Use case 1: Related Works

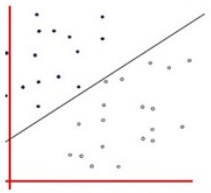
- Significant work done in the field of quality prediction.
- As there are several papers, we introduce a categorization to better summarize the main contributions.
- Two approaches followed so far:
  - Standard statistical techniques used in earlier studies;
  - Machine learning techniques used in later ones.


Discriminant Analysis


Artificial Neural Network


Logistic regression


Decision Trees


Support Vector Machine

Use case 1: Related Works

In both approaches

- the validation phase was unsatisfactory;
- difficult to find a technique valid for every software project.

With respect to the past

- Our model takes as input not only metrics but also the mathematical modeling of best practices as described in the CMMI specification.
- Our model takes as input all the metrics not just the object-oriented ones.

Use case 1: What has been done?

- Planned the validation of our model with a progressive increase in the data set to properly speculate on the variables included in the model.
- Evaluated existing metrics tools, such as CLOC, SLOCCount, Metrics, Pmccabe and Understand.
- Collected data about product metrics from EMI1 to EMI3 distributions.
- Used a Matlab-based prototype tool that codes the presented solution.
- Determined the level of risk to contain faults for each software component and then grouped for software products
- Identified the importance of the metric amongst those considered.

Use case 2

- To guarantee the maintainability of software used in scientific researches where critical use cases are addressed, as in the context of shielding and radiation protection:
  - By using existing standards that identifies the software characteristics;
  - By exploiting a set of product metrics that allows understanding the code state.

Use case 2

- The software selected is Geant4.
- It is the most cited paper in Nuclear Science Technology;
- It is used in a wide variety of scientific contexts, such as shielding and radiation protection;
- It is developed and maintained by an international widespread collaboration;
- It is a mature system (20 years old).
- It is a playground to study metrics and metrics tools.
- Can metrics help addressing its maintainability in the next 20 years?

Use case 2: preliminary scope of the analysis

Initial appraisals concern a subset of Geant4 packages with a key role in scientific applications:

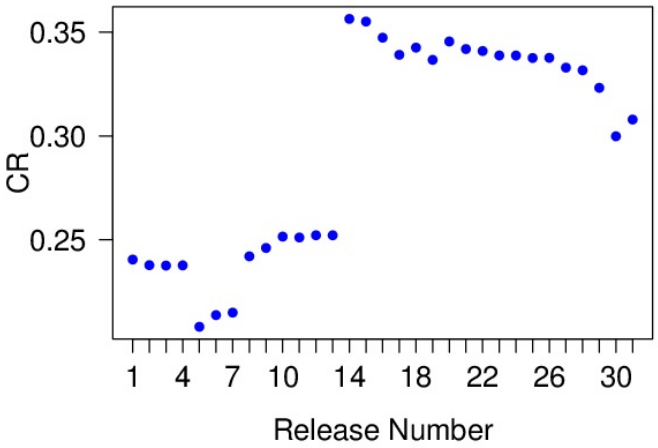
- the Geometry package makes it possible to describe a geometrical structure and navigate through it;
- the Processes package handles physics interactions;
- the PhysicsLists package contains physics selections.

Use case 2: Geant4 releases over time

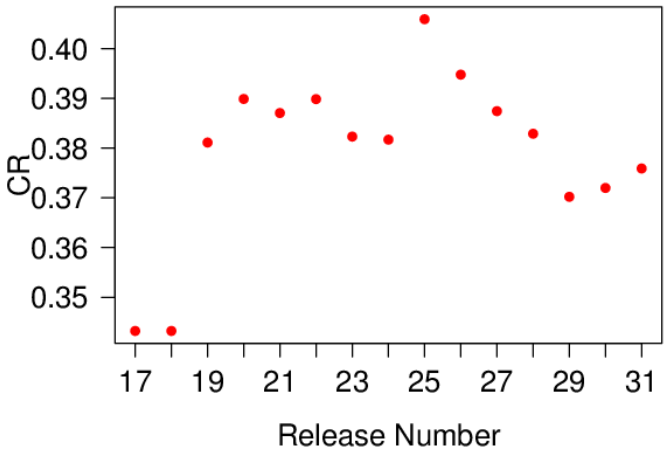
Number	Name	Year	Number	Name	Year	Number	Name	Year
1	0.0.p04	1999	11	5.0.p01	2003	21	8.2.p01	2007
2	0.1	1999	12	5.1.p01	2003	22	8.3.p02	2008
3	1.0	1999	13	5.2.p02	2003	23	9.0.p02	2008
4	1.1	2000	14	6.0.p01	2004	24	9.1.p03	2008
5	2.0.p01	2000	15	6.1	2004	25	9.2.p04	2010
6	3.0	2000	16	6.2.p02	2004	26	9.3.p02	2010
7	3.1	2001	17	7.0.p01	2005	27	9.4.p04	2012
8	3.2	2001	18	7.1.p01	2005	28	9.5.p02	2012
9	4.0.p02	2002	19	8.0.p01	2006	29	9.6.p04	2015
10	4.1.p01	2002	20	8.1.p02	2006	30	10.00.p04	2015
						31	10.01.p01	2015

Use case 2: some measurements over release

Geometry



PhysicsLists

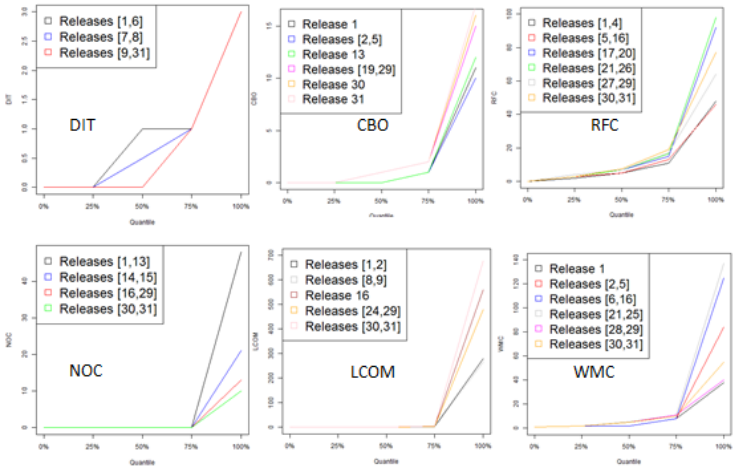


Size category at file group

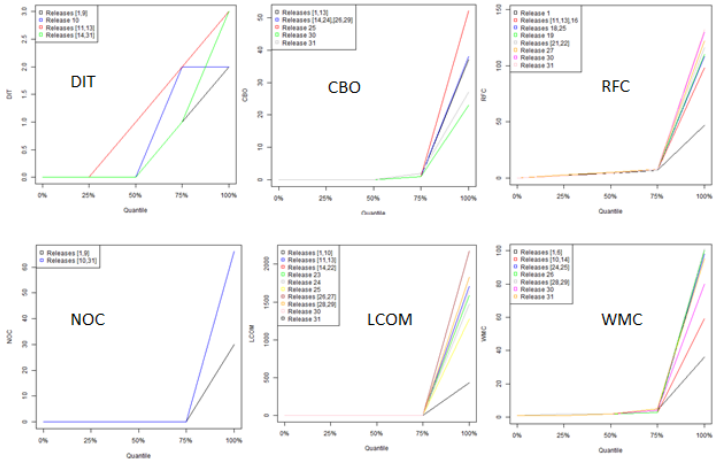
Package	N. Files	TLOC	CLOC	SLOC
Geometry	[501, 697]	near 5 million	[32K, 59K]	[50K, 116K]
Processes	[840, 3492]	13 million	[25K, 222K]	[138K, 469K]
PhysicsLists	[203, 410]	430 thousand	[6K, 17K]	[8K, 23K]

Use case 2: some measurements over release

Geometry



Processes

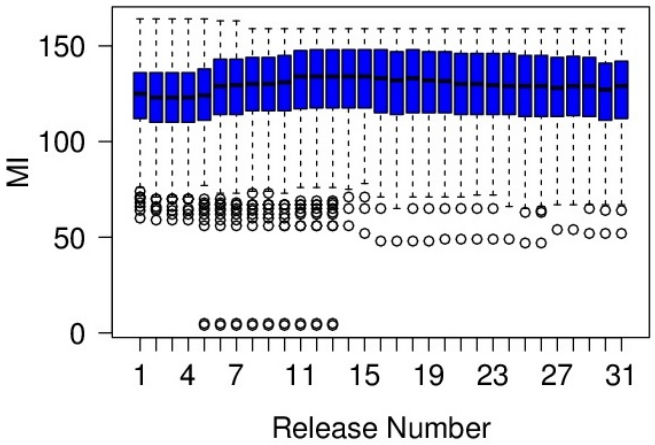


Object-Oriented category at class group

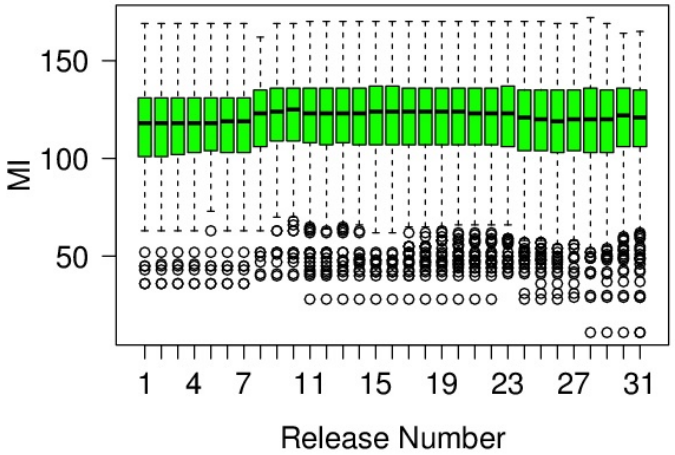
Package	DIT	NOC	CBO	LCOM	RFC	WMC
Geometry	3	[10,18]	[10,17]	[266,677]	[46,98]	[38,137]
Procesess	[2,3]	[30,66]	[23,52]	[435,2168]	[47,133]	[36,101]
PhysicsLists	1	[10,18]	[6,14]	[99,114]	[61,80]	[57,65]

Use case 2: some measurements over release

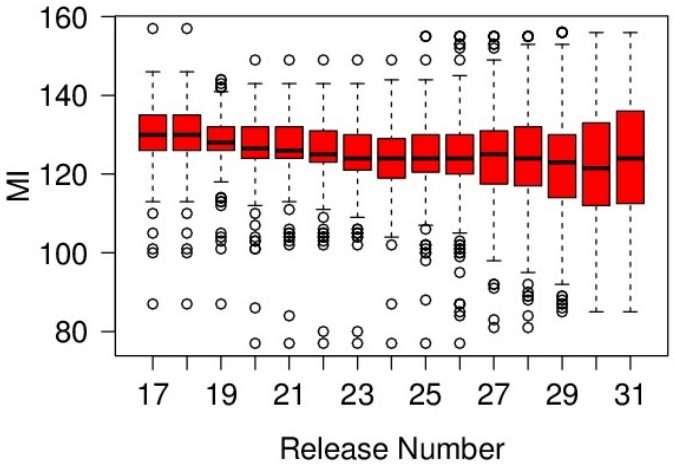
MI at file group for Geometry



MI at file group for Processes



MI at file group for PhysicsLists



Complexity category at class group

Package	MMCC	Package	MMCC
Geometry	[11,31]	PhysicsLists	[19,47]
Procesess	[18,33]		

Investigation on thresholds

Open Issues

- Thresholds of software goodness documented in the literature derive from specific domains, such as aerospace, telecommunication and student exercises
- Thresholds values are quite old
  - therefore they may not reflect the evolution of the programming languages
- They may reflect domain-specific software characteristics
  - therefore they cannot be blindly applied to our field

Investigation on thresholds

What thresholds should we consider?

Literature can help identifying limitations of current thresholds definition, such as those derived from experience in specific domains, error models and cluster techniques.

Research work

Identifying suitable thresholds for HEP software

Sum up: what have we done?

- Identified and evaluated existing metrics tools
- Identified a set of product metrics for analysing the maintainability software characteristics
- Built a dataset of product metrics to investigate the quality of a set of EMI software and a set of Geant4 packages
- Performed a predictive fault-component analysis for EMI packages, such as StoRM, VOMS, WNoDeS, WMS and YAIM [1]
- Started analysing data for Geometry, PhysicsLists and Processes packages of Geant4 from 0 up to 10 releases [2, 3]
- Developed Matlab and R scripts to analyse data

Sum up: as initial assessment

- The predictive analysis applied to some EMI packages provides details about fault-prone and non fault-prone software components
- The quantitative data about the selected metrics for Geant4 provides information about software characteristics and trends by highlighting part of Geant4 that would benefit from close attention regarding future maintainability

Future plans

Operative activities

- Include software metrics tools and static analysis tools in the continuous integration infrastructure based on Jenkins
  - already performed some tests with the Understand plugin
  - included clang, parasoft c++test and jtest in Docker images
- Propose workaround to those pieces of software that show a lower quality level
- Identify and evaluate open metrics tools for Java programming languages

Future plans

Research activities

- Determine which product metrics are most effective at identifying risks for Geant4
- Work on the identification of appropriate thresholds and ranges
- Extract files, classes and functions of Geant4 that contains outliers according to statistical analysis
- Analyse and employ further statistical techniques in addition to discriminant analysis and regression in the proposed quality model

Acknowledgements

- Marco Canaparo for collaborating with the development of software quality model and the related challenges
- Maria Grazia Pia and Francesco Giacomini for supporting the Geant4 research
- Davide Salomoni for supporting this activity
- Cinzia Viroli (Associate Professor in Statistics at Department of Statistical Sciences "Paolo Fortunati", Bologna University) for providing statistical and R feedback

References

1. Elisabetta Ronchieri, Marco Canaparo, Davide Salomoni, "A software quality model by using discriminant analysis predictive technique," In Transaction of the SDPS: Journal of Integrated Design and Process Science, 2014
2. Elisabetta Ronchieri, Maria Grazia Pia, Francesco Giacomini, "Software Quality Metrics for Geant4: An initial assessment," In the Proceedings of ANS RPSD 2014, Knoxville, TN, September 14 – 18, 2014
3. Elisabetta Ronchieri, Maria Grazia Pia, Francesco Giacomini, "First statistical analysis of Geant4 quality software metrics," under review for CHEP2015 proceedings